NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

CLASS TRANSLATOR FOR THE FEDERATION INTEROPERABILITY OBJECT MODEL (FIOM)

by

Lee, Shong Cheng

March 2002

Thesis Advisor:	Valdis Berzins
Thesis Co-Advisor:	Luqi
Second Reader:	Paul Young

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2002	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Class Translator for the Federation Interoperability Object Model (FIOM)	**5. FUNDING NUMBERS** **ARO** **DMSO** **NAVSEA**
6. AUTHOR(S) Lee, Shong Cheng	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

There is a growing need for systems to inter-operate in order to facilitate information sharing and to achieve objectives through joint task executions. The differences in data representation between the systems greatly complicate the task of achieving interoperability between them. Young's Object Oriented Method for Interoperability (OOMI) defines an architecture and suite of tools to resolve representational differences between systems. The OOMI architecture and tool suite will reduce the labor-intensity and complexity of the integration of disparate systems into a cooperative system of systems (federation of systems) and their subsequent deployment. At the heart of this architecture is the definition of translations between any two different classes of objects and a run-time component (the Translator) that will execute such translations.

This thesis describes a prototype framework that implements the OOMI, a prototype class translation code generator that assists an Interoperability Engineer in the definition of the translations and a prototype Translator that executes these translations.

14. SUBJECT TERMS Command, Control, and Communications, Computing and Software			**15. NUMBER OF PAGES** 355
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

CLASS TRANSLATOR FOR THE FEDERATION INTEROPERABILITY OBJECT MODEL (FIOM)

Lee, Shong Cheng
Defence Science and Technology Agency, Singapore
B.Sc. (Hons), National University of Singapore, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2002

Author:	Lee, Shong Cheng
Approved by:	Valdis Berzins Thesis Advisor
	Luqi Thesis Co-Advisor
	Paul Young Second Reader
Approved by:	Chris Eagle Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

There is a growing need for systems to inter-operate in order to facilitate information sharing and to achieve objectives through joint task executions. The differences in data representation between the systems greatly complicate the task of achieving interoperability between them. Young's Object Oriented Method for Interoperability (OOMI) defines an architecture and suite of tools to resolve representational differences between systems. The OOMI architecture and tool suite will reduce the labor-intensity and complexity of the integration of disparate systems into a cooperative system of systems (federation of systems) and their subsequent deployment. At the heart of this architecture is the definition of translations between any two different classes of objects and a run-time component (the *Translator*) that will execute such translations.

This thesis describes a prototype framework that implements the OOMI, a prototype class translation code generator that assists an Interoperability Engineer in the definition of the translations and a prototype *Translator* that executes these translations.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF SYMBOLS, ACRONYMS, AND/OR ABBREVIATIONS

CCR	Component Class Representation
DTD	Data Type Definition
FCR	Federation Class Representation
FE	Federation Entity
FEV	Federation Entity View
FIOM	Federation Interoperability Object Model
SoS	System of Systems
GUI	Graphical User Interface
IE	Interoperability Engineer
IDE	Integrated Development Environment
MGRS	Military Grid Reference System
OOAD	Object-Oriented Analysis and Design
OOMI	Object Oriented Method for Interoperability
RWE	Real-World Entity
SAX	Simple API for XML
UML	Unified Modeling Language
URI	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Mark-up Language
XPath	XML Path Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Capt Young for his timely and insightful reviews on the writing of this thesis.

-- LEE, Shong Cheng

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. OBJECTIVE

The Object Oriented Method for Interoperability (OOMI) [Young02] defines an object-oriented method and a suite of tools to resolve representational differences between systems. The OOMI method and tool suite will reduce the labor-intensity and complexity of the integration of disparate systems into a cooperative system of systems (federation of systems) and their subsequent deployment.

The objective of this thesis is to develop a prototype implementation of the translation generator module for the OOMI IDE and a prototype translator that will make use of the translations created by the IDE.

The OOMI and the OOMI IDE will be introduced in detail under section II.B.

B. THE INTEROPERABILITY PROBLEMS

Few systems operate in a totally isolated environment where they are fully self-contained, with no requirement to interoperate with other systems. Many systems interoperate with each other through the exchange of information and the joint execution of tasks.

Legacy systems are often enhanced with capabilities to export their data or to make their operations available to other systems in order to achieve new objectives or enhance performance. New systems that consume information from others and produce information for external consumption are being developed and coming on-line everyday. Similarly, both legacy and new systems typically have requirements to interoperate with other systems through the joint execution of tasks in order to achieve their objectives.

Such trends in system integration led to the new concept in information technology: development of a “System of Systems (SoS)”, also commonly known as a “Federation of systems”. A system of systems is a composite system made up of independently developed systems. We should also take note that almost all non-trivial software systems are composed of smaller subsystems or components. In a way, all systems can be considered as a system of systems. In order to interoperate, the component systems in the system of systems exchange information and operations used to model the real-world objects (also known as real-world entities from the component system's problem domain). This implies that each component system needs to understand the models used by other systems to represent the real-world entities involved in system interoperation. This leads to the *$O(n^2)$ translations* problem.

The *$O(n^2)$* translations problem occurs when legacy systems are being integrated due to the potentially different models used by the independently developed systems. This is also a problem for new systems that are being developed, if they are required to interoperate with other autonomously developed systems during their lifetimes.

1. The $O(n^2)$ Translations Problem

Systems of systems (SoS) are rapidly being developed to meet the stringent demands of today's applications. A system of systems may be composed not only of legacy systems, but also newly developed systems. As new systems are being developed, each will have to consume data from other systems in diverse formats.

A producer typically exports data to more than one consumer. The consumers may or may not agree on a common data format with the producer. Similarly, a consumer may need to accept data from multiple producers; each producer may export its data in a unique format. As a result, each producer has to export data in multiple formats, while each consumer has to be able to process data in different formats for each producer it communicates with. Each system has to translate the incoming or outgoing data as required, resulting in a worse case scenario of having to have $n(n-1)$ translations in a SoS comprising n systems. That is, a worse case complexity of $O(n^2)$ is experienced in terms of the number of translations for the systems within a SoS, as shown in Figure I-1.

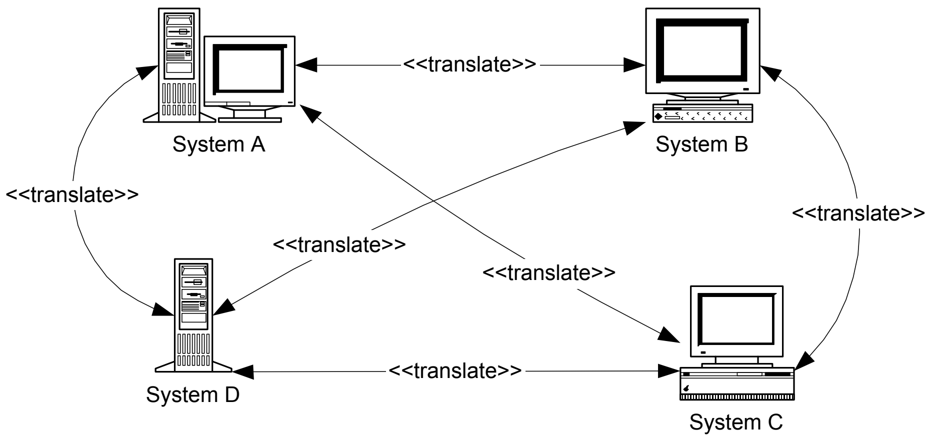


Figure I-1. The O(n^2) translations problem.

2. Computer-Aid In Managing Common Representations

The most obvious solution to the *O(n^2)* translations problem would be for all systems that exchange information about real-world entities to agree on a common model for the shared information. Information can then be exchanged using the common model, with each system only concerned with the translation between the common model and its own internal model, as illustrated in Figure I-2.

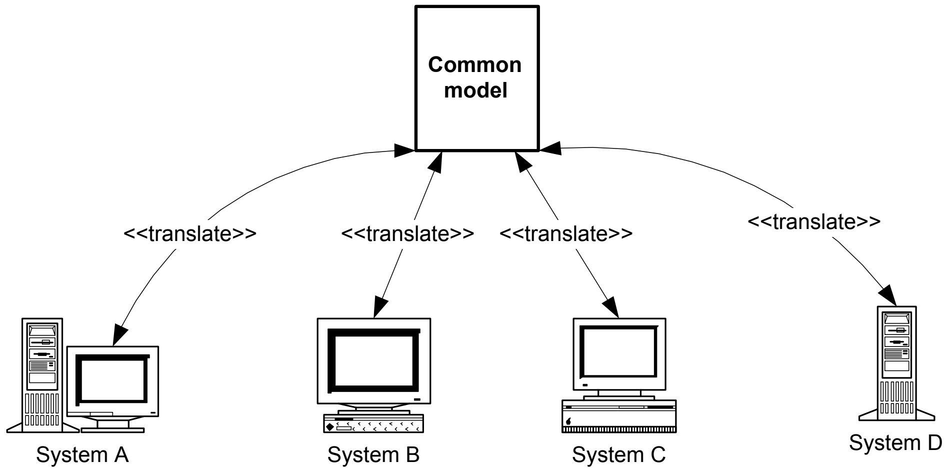


Figure I-2. Multiple systems sharing a common model.

As the component systems evolve and as new systems join the federation, the common model will evolve. As growth in the complexity of the common representations is inevitable, a tool to aid in the construction, organization and subsequent maintenance of the set of common representations is needed.

The OOMI and its set of tools, described in more detail in chapter II, are designed to address the *$O(n^2)$* translations problem, to provide an object-oriented method for constructing a set of common representations and to provide computer-aid in its construction, organization and subsequent maintenance.

II. BACKGROUND

A. HETEROGENEITY

1. Eight Classes of Heterogeneity

Interoperating systems share information about real world objects, but such information is typically represented differently. Gio Wiederhold, a pioneer in early multidatabase efforts, defined seven classes of heterogeneity. Based on Wiederhold's work, Young added *heterogeneity of structure* and renamed *heterogeneity of representation* to *heterogeneity of denotation*, resulting in the following classification of modeling differences between different systems [Young02]:

1. *Heterogeneity of Hardware and Operating Systems*
2. *Heterogeneity of Organizational Models*
3. *Heterogeneity of Structure*
4. *Heterogeneity of Denotation*
5. *Heterogeneity of Meaning*
6. *Heterogeneity of Scope*
7. *Heterogeneity of Level of Abstraction*
8. *Heterogeneity of Temporal Validity*

A more detailed discussion, summarized from [Young02], of the eight classes of heterogeneity follows:

a. Heterogeneity of Hardware and Operating Systems

Hardware platform and operating system differences can result in differences in the physical representation of information. For example, in the representation of integers, differences in word order such as Little Endian versus Big Endian are common between systems. Such low-level differences are generally encapsulated from the application developers through the use of distributed computing frameworks such as the Common Object

Request Broker Architecture (CORBA), Microsoft’s .Net, Distributed Component Object Model (DCOM) or Java’s Remote Method Invocation (RMI).

b. Heterogeneity of Organizational Models

Heterogeneity of organizational models refers to differences in the conceptual models used by autonomously developed systems. In the context of interoperability, heterogeneity of organizational models can refer to differences in analysis and design principles employed, such as use of an Object-Oriented Analysis and Design (OOAD) approach versus a structured analysis approach.

c. Heterogeneity of Structure

Variations in the structure of how information is arranged can occur among systems using the same organizational model. These variations can include differences in structural composition, possible schema mismatches, and variations due to the presence of implied information. For example, a customer may be modeled as an object in one system, but in another system, it may be modeled as a string containing the unique identification of the customer.

d. Heterogeneity of Denotation

Heterogeneity of denotation includes:

- domain mismatch problems – occur when the same concept is characterized differently, for example, geographic position can be measured in longitude/latitude in one system, and be measured in Military Grid Reference System (MGRS) in another system
- differences in units of measurements – for example, one system may use the imperial units of measurement, while other systems may be using the metric system, such differences may occur even within the same system, between different autonomously developed modules
- differences in precision – for example, one system may use a single precision floating point representation, while another system may use a double precision floating point representation

- disparate data types – for example, date may be represented as a record containing the integer fields year, month and day, or as a character string like “1 January 1970”
- different field lengths – for example, different field lengths may be allocated for the name of a person on different systems
- different integrity constraints on the same value – for example, one system might allow a value for missile effectiveness in the range of one to twenty nautical miles whereas a different system might allow effective range values up to thirty nautical miles

e. Heterogeneity of Meaning

The use of synonyms, homonyms and abbreviations contribute to this kind of difference. For example, synonyms like location and position can both be used to describe geographic coordinates. The same word can also have different meanings (homonyms), for instance, the word tank can describe both a tracked combat vehicle and a water container. The use of abbreviations is a special case of the use of synonyms. Different abbreviations can represent the same entity, such as the use of POSIT and POS to refer to position.

f. Heterogeneity of Scope

Heterogeneity of scope results from differences in the information used to model a real world entity. Different applications will capture different attributes of a real world entity, based on their specific needs. For example, a logistics management system might include attributes fuelCapacity and ammunitionStatus for a main battle tank, whereas a command and control system would include attributes weaponRange and defensiveArmor in its tank model.

g. Heterogeneity of Level of Abstraction

Different systems use different levels of abstraction when modeling a real world entity. For example, a division level command and control system models an infantry platoon as a single entity, while a battalion level command and control system may model a platoon as a collection of sections or groups.

h. Heterogeneity of Temporal Validity

Modeling differences may result from differences in when the model observes or records the state of a real-world entity and differences in the length of time a piece of data remains valid. For example, a battalion level command and control system may require a second-by-second update on the positions of all its platoons, a division level command and control system only needs a minute-by-minute update.

2. Differences in View and Representation

The eight classes of heterogeneity are further categorized by Young as differences in view and differences in representation.

Differences in view: Different systems capture different characteristics of a real world object (*heterogeneities of scope, level of abstraction* and *temporal validity* [Wie93,Young02]). For example, in Figure II-1, three different views of a surface-to-surface missile system are provided by the systems modeling that real-world entity. In the first view, systems A and D capture information about the entity's *type*, *position* and *time*, whereas system B provides a model of its *position*, *time* and *range*. Finally, system C provides a third view of the surface-to-surface missile capturing the entity's *type*, *position*, *time* and *range*.

Differences in representation: In addition, the same characteristics may be represented differently (*heterogeneities of organizational models, structure* and *denotation* [Wie93, Young02]). For instance, as shown in Figure II-1, a specific location on the globe may be represented by latitude and longitude (system A *position*) on one system, or be represented by the military grid reference system (system B *location*) on another.

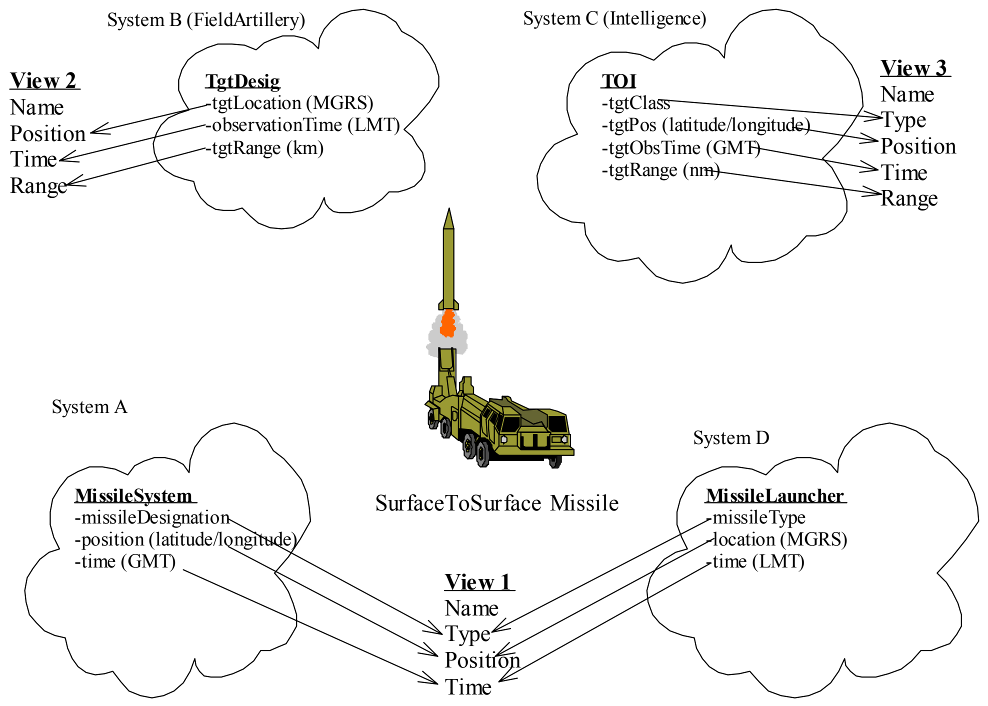


Figure II-1. Differing Real-World Views of the same object [from Young01].

B. OBJECT-ORIENTED METHOD FOR INTEROPERABILITY (OOMI)

1. Introduction

The Object-Oriented Method for Interoperability (OOMI) [Young02] proposed by Young is a methodology that addresses the $O(n^2)$ translations problem in interoperability and is a systematic method in which information shared between interoperating systems can be modeled in a standard, object-oriented manner.

The OOMI consists of three major components as shown in Figure II-2:

- The Federation Interoperability Object Model (FIOM) – the standard model shared between systems in a federation.
- The OOMI Integrated Development Environment (OOMI IDE) – a graphical user-interface based development environment used to maintain the FIOM.

- The OOMI Translator – the run-time component that will resolve the differences in information models of component systems in a federation.

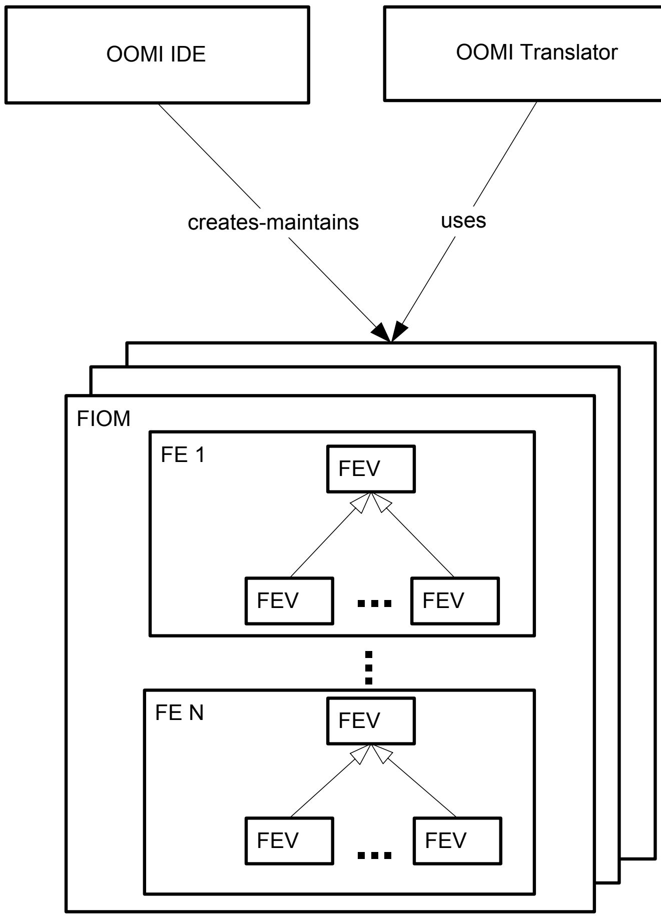


Figure II-2. Major components of OOMI.

2. The FIOM

The OOMI proposes that inter-operating component systems be organized into a federation with a common Federation Interoperability Object Model (FIOM) containing information on the Real-World Entities (RWEs) shared among federation components. The FIOM (Figure II-3) is composed of Federation Entities (FEs), each representing a real world entity. The FE comprises a hierarchical tree of Federation Entity Views (FEVs) forming an inheritance hierarchy of FEVs within the FE. Each FEV represents the view that a component system or group of component systems has of the RWE.

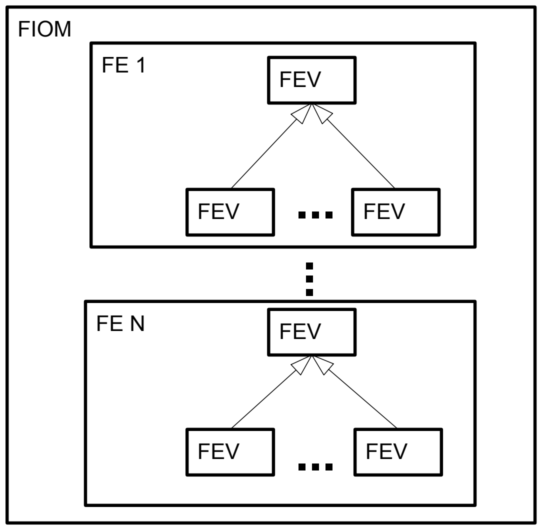


Figure II-3. The FIOM.

Each FEV contains exactly one Federation Class Representation (FCR) which represents the standard representation of the view. The FCR serves as the intermediate representation for translation between the source and destination systems. An FEV also contains one or more Component Class Representations (CCRs). The CCR is a direct mapping of a component system's unique representation of the RWE. Translations between the FCR and CCR are defined within the FIOM. Figure II-4 depicts an FEV's components.

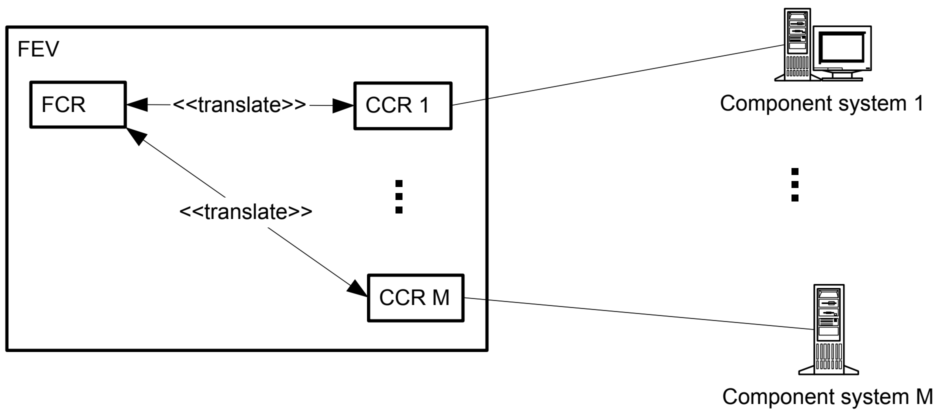


Figure II-4. FEV, FCR, CCR and Component Systems.

Conceptually, a FCR or CCR contains a set of properties (attributes and operations) that represents the state and behavior of the RWE, as depicted in Figure II-5.

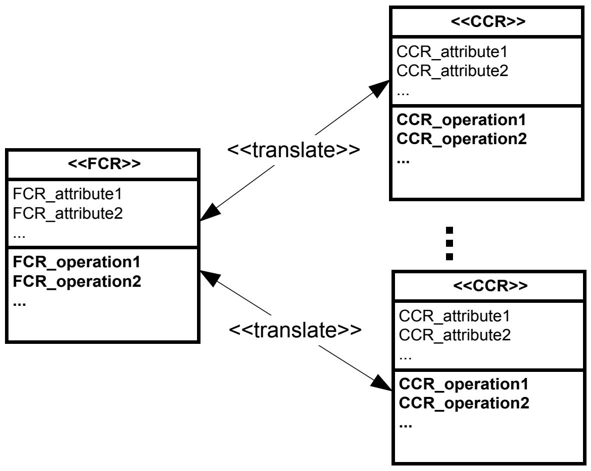


Figure II-5. Conceptual views of FCR and CCR.

Since an FEV is represented by exactly one FCR, an FEV can be visualized as a class with a set of attributes and operations, as illustrated in Figure II-6.

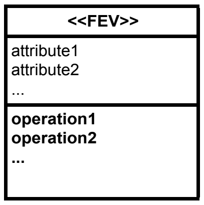


Figure II-6 A simplified view of an FEV.

Using the FCRs as the standard representation for RWEs, each component system only has to provide the logic to translate between its unique representation of the RWE and that of the corresponding FEV.

One important property that must be maintained within an FE is the following:

For any federation entity, E, and X, Y, where X and Y are different views (FEV) of E; $\rho(X)$ and $\rho(Y)$ are the set of properties of X and Y respectively. If there exists an attribute named α, such that $\alpha \in \rho(X) \wedge \alpha \in \rho(Y)$, then α must have the same real world meaning within the context of E in both X and Y, and the value of α can be directly copied between α of X and α of Y.

This property is exploited in the OOMI translator to automatically determine candidate destination FCRs for a source FCR.

As shown in Figure II-7, the number of unique translations required within a federation comprising *n* systems using the OOMI is *2n* or *O(n)*, instead of *O(n^2)*, as in the case without a FIOM.

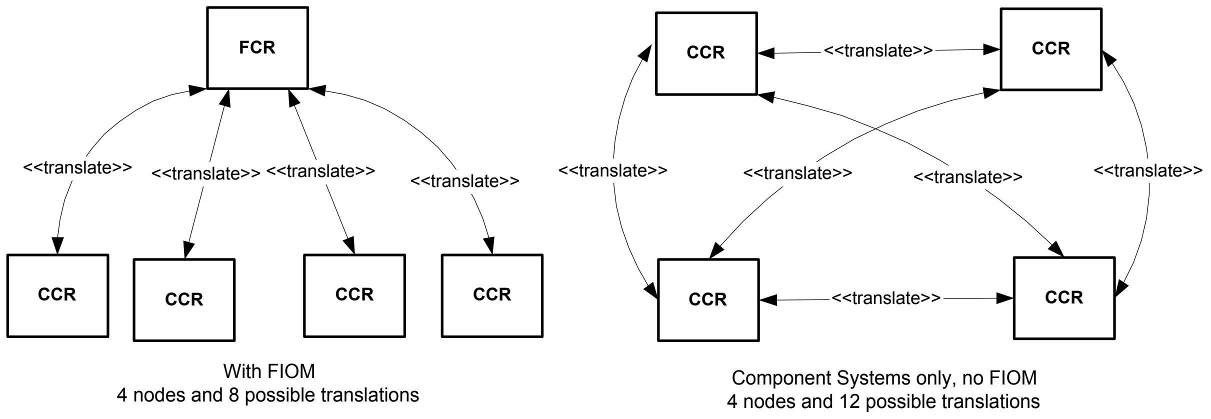


Figure II-7. Translations required with and without FIOM.

The FIOM also provides benefits in terms of the visibility and understandability of the shared information and a foundation for easy extension as new systems are added to an existing federation [Young01].

3. The OOMI IDE

The OOMI IDE is a graphical user interface (GUI) based tool that is used by the Interoperability Engineer (IE) to construct and maintain FIOMs. The main functions of the OOMI IDE allows the IE to perform the following:

- Maintain multiple FIOMs
- Specify different views of a real-world entity resulting from the different perspectives each component system has of that entity
- Construct and maintain an inheritance hierarchy relating the different views of a real-world entity
- Define standard federation representations of the real-world entity views identified and establish the relationship between the standard view representation (FEV and its defining FCR) and the various component system view representations (CCR)

- Define the transformations (also known as translations) required to translate between component system and standard representations of a view

4. The OOMI Translator

The OOMI Translator uses the FIOM, which the IE constructed using the OOMI IDE, to reconcile differences in real-world entity view and representation among component systems of a federation at run-time. The design and implementation of the prototype translator is presented in more detail in chapter V.

C. OTHER METHODS FOR INTEROPERABILITY

Interoperability between systems takes two basic forms:

1. Exchange of information
2. Remote procedure/method/service invocations

In this section, we will briefly look at the tools currently available to achieve interoperability.

1. Distributed Computing Frameworks

Distributed computing frameworks encapsulate the ugly details of lower level networking application programming interfaces (API) like the popular Berkeley Sockets (BSD) API. Currently, there are two basic types of distributed computing framework: non-object based and object based.

The Remote Procedure Call (RPC) protocol is a non-object-based framework. The RPC protocol provides encapsulation over the differences in representation of primitive data types (byte-order, number of bits, etc.) due to the underlying hardware, operating systems and programming languages.

Object-based frameworks provide a higher level of abstraction than RPC by introducing the concept of remote objects and remote invocations of these objects' methods. These frameworks extend the object-oriented programming paradigm over the network, although in general, inheritance is not support in the object-based frameworks. Most implementations of these frameworks also feature a Broker architecture for service discovery, location transparency and load balancing. Examples of such frameworks are the Common Object Request Broker Architecture (CORBA), Microsoft's .Net, Distributed Component Object Model (DCOM) or Java Remote Method Invocation (RMI).

Distributed computing frameworks do not address the *$O(n^2)$ translations* problem, as they do not define a methodology for systematically developing a common representation for information between interoperating systems. But they are useful in serving as the underlying vehicle for network communications.

2. BizTalk

BizTalk is a Microsoft initiative to address document interchange between businesses. It is a specification for business-to-business exchanges using XML-based SOAP 1.1 messages. BizTalk is an "open" standard, supported by Boeing, Compaq, SAP, United Parcel Service (UPS) and other companies.

BizTalk comprises three major components:

- BizTalk Framework Independent Document Specification
- BizTalk.org
- BizTalk Framework Compliant (BFC) Server

Conceptually, the BizTalk framework is similar to the OOMI in the sense that a standard set of XML document schemas will be defined and published for use. The BizTalk Web site will house schemas (or "contracts") that developers using XML and BizTalk have proven effective in specific, vertical implementations with specific software packages. To be posted publicly, the schemas must pass an automated standard conformance script (promised to be freely viewable). These schemas dictate "what I send you, what you send me back, what you promise, what you respond," and even what specific documents are sent back and forth [Hadfield].

The BizTalk framework requires that the participating systems conform to the published standard schemas. It does not provide tools to resolve the representational differences between the standard schemas and legacy systems' native data format. Such tools are to be provided by third party vendors, for example, the Microsoft BizTalk Server 2002. These third party tools allows a 2-step transformation process that resolves the *$O(n^2)$ translations* problem. But these tools do not provide models similar to the OOMI's FIOM to capture differences in views of real world entities in a way that allows easy extension.

3. Generic Interoperability Framework (GINF)

> "The goal of the Generic Interoperability Framework (GINF) is to facilitate interoperability between heterogeneous systems. GINF is a set of principles, which describe an application-neutral way of interaction between software components. The key principles include:
>
> -- Generic representation: protocol information, languages, data and interface descriptions are represented in a uniform manner using directed-labeled graphs.
>
> -- The ability to dynamically fetch a machine-readable description for every piece of information exchanged between components."
>
> (from [Melnik+])

The Generic Interoperability Framework (GINF) proposes that messages exchanged between systems be based on the Resource Description Framework (RDF) models. These models are directed labeled graphs where an edge of the graph represents a predicate that holds between a subject node and an object node. Nodes can be either a resource node or a literal node. Resources are entities that can be specified using a Uniform Resource Identifier (URI). String or binary data are described using literals. On receiving a message (a graph), the system can look up schema information that describes unknown elements used in the graph using the URIs specified. [Melnik+99]

The key here is that a component system is able to dynamically discover the meanings of the elements in the document received using their associated URIs. It does not address the *$O(n^2)$ translations* problem, since, for a system to talk to N other systems, it will still have to accept their documents, each possibly represented in N ways, and be able to translate such documents into a form suitable for its own processing. However, such translation is simplified due to the semi-parsed format of the directed labeled graph model of the RDF specification.

D. OVERVIEW OF XML AND JAVA-XML DATA-BINDING

The initial prototype of the OOMI toolset is implemented in Java. The prototype requires all component systems to export or import their information model as well-formed XML documents. These XML documents are transformed into Java objects using Java-XML Data-Binding.

1. XML

The Extensible Markup Language (XML), which is a meta-markup language that became a World Wide Web Consortium (W3C) recommendation in January of 1999, is the language of choice for structured documents and data on the Internet. It is platform-independent, non-proprietary, customizable, self-describing and human readable (human readable, as opposed to proprietary binary formats). These properties make it desirable for data representation in almost any system. However, it should also be noted that XML might not be suitable in environments where processing power, memory or bandwidth is limited, as in embedded systems or low-end mobile devices.

2. JAVA-XML DATA-BINDING

Java-XML Data-Binding is the process in which an XML document is converted to its equivalent representation, as a Java object, and vice versa. The Java™ Architecture for XML Binding (JAXB) Working Draft Specification [Sun01] defines XML data-binding as a facility containing two components: A *schema compiler* and a *marshaling framework*. As shown in Figure II-8, the *schema compiler* reads an XML schema to derive a set of Java classes representing the structure of the XML documents that conform to the XML schema. During runtime, the *marshaling framework* unmarshal an input XML document into instances of the derived classes. The member variables of these derived classes provide access to the content of the input XML document's elements.

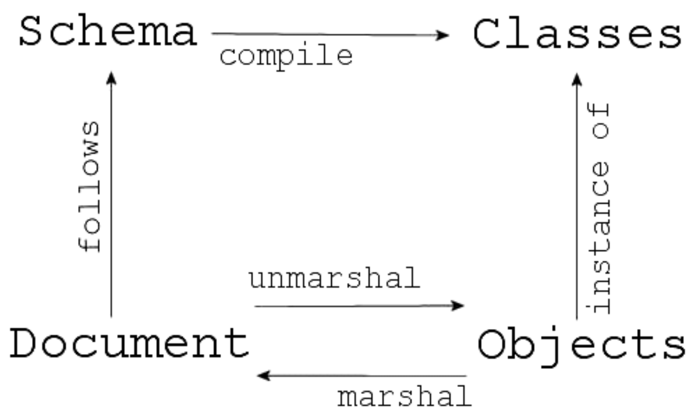


Figure II-8. XML and Java Relationships (from [Sun01]).

Java-XML Data-Binding makes XML easy to use by compiling an XML schema into one or more Java technology classes. The marshaling framework and the derived classes handle all the details of XML parsing and formatting. Similarly, the generated classes ensure that the constraints expressed in the schema are enforced in the resulting methods and Java technology language data types.

E. SUMMARY

This chapter provided an introduction to the OOMI, which is a methodology for resolving modeling differences between systems, and the OOMI toolset providing computer aid to the process. The next chapter will discuss the requirements of the OOMI IDE Translation Generator and the OOMI Translator. The FIOM framework, which is the foundation of the OOMI IDE and the OOMI Translator, is also presented.

III. RESOLVING MODELING DIFFERENCES AMONG SYSTEMS USING OOMI

A. INTRODUCTION

For the OOMI IDE Translation Generator and OOMI Translator prototype, Java is the implementation language while XML is used as the on-the-wire format for information exchange between the OOMI Translator and the component systems. Each component system is expected to have its own XML schemas that describe the information and operations it exports or imports. XML documents conforming to these schemas will be transferred between the OOMI Translator and the component systems to accomplish information exchange and joint task execution. The OOMI Translator is responsible for invoking the translations to perform the appropriate transformations required for each component system using the information.

Figure III-1 illustrates how an XML document exported from Component System A is translated to the various formats required by Component Systems 1 to N by the OOMI-Translator. The XML documents being transmitted represent the different views of the same real world entity that is used by the component systems.

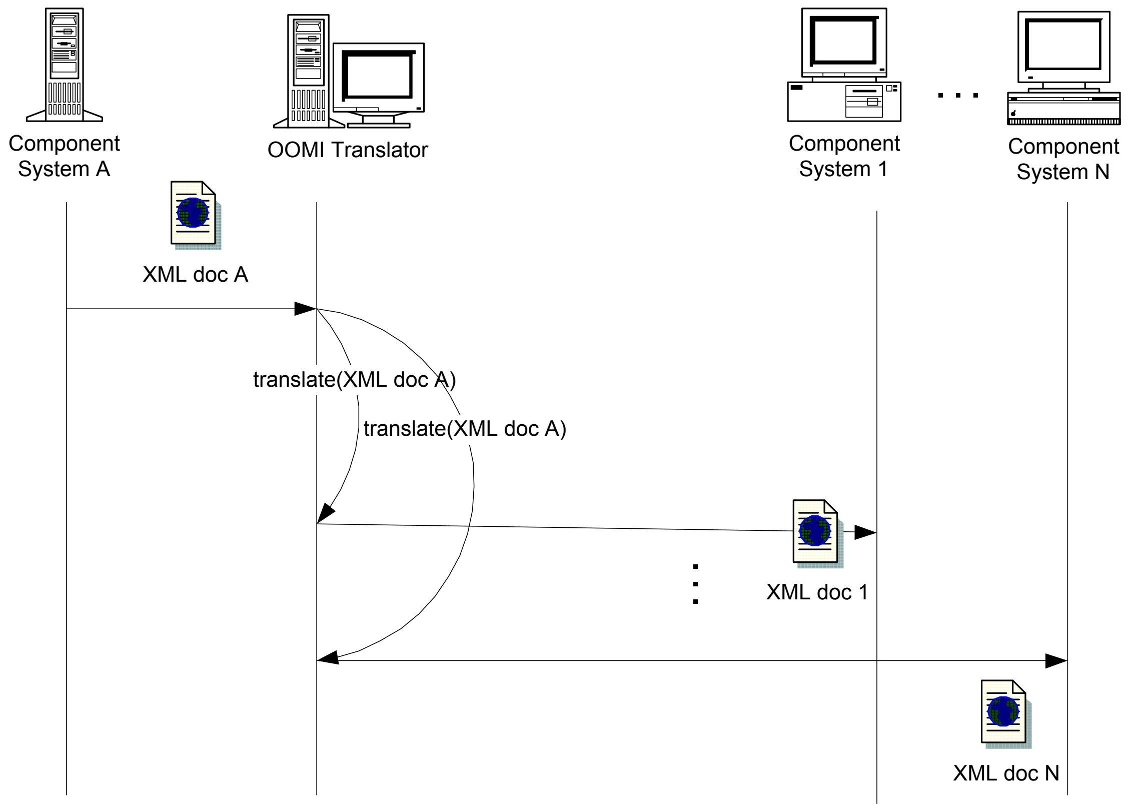


Figure III-1. OOMI operation overview.

Internally, the OOMI Translator converts, or unmarshals, the XML documents it receives into their equivalent Java objects (instances of CCR), through XML data-binding. The CCR instances (source CCR instances) are then translated to their corresponding standard FCR representation (also Java objects). For each destination system, the translator object, instantiated by the OOMI Translator, will resolve differences in view between source and destination systems, converting the source FCR instance for the destination system into a destination FCR instance. This destination FCR instance is then translated into a corresponding CCR instance for the destination system. The logic behind the translations between an FCR instance and a CCR instance is defined by the interoperability engineer through the OOMI IDE's Translation Generator module. Finally, the destination CCR instance is converted, or unmarshalled, into the XML document expected by the destination system. This process is further elaborated in section V.C. A detailed description of the Translation Generator is presented in Chapter IV, while the details of the OOMI Translator can be found in Chapter V.

XML had been chosen for the prototype implementation of the OOMI due to the following:

1. XML is the most widely adopted standard for information exchange on the Internet.
2. Due to the above, there is availability of a large number of tools and software libraries (both commercial-off-the-shelf and open-source) in support of XML and its related technologies. They are also available on most operating systems.
3. DoD will be standardizing on XML for information exchange between DoD systems [CY01].
4. It is reasonable to assume that most systems, if not all, that export data to other systems and/or import data from other systems will prefer XML, due to all of the above reasons.

The Java language is chosen as the language to implement the translations between XML documents rather than using XSLT (Extensible Style Language Transformation) due to the following benefits Java has over XSLT:

1. Strongly typed
2. Better readability
3. Better scalability
4. Ease of maintenance
5. Full capabilities of a general purpose programming language: database access, network, access to distributed components, etc
6. Extensible to non-XML based object transfer, XSLT can only be applied to XML documents

The last benefit is perhaps the most important, as the OOMI is not dependent on XML to achieve its objectives, it can be applied to any information exchange format equally well. One of the main design considerations in the development of this prototype is to ensure that extending it to work on non-XML documents can be easily achieved, simply by using a marshalling framework that marshals between raw binary data and Java objects. A good example would be the object serialization mechanism provided by the Java run-time environment.

Two major components of the OOMI are presented in chapters IV and V, namely, the the Translation Generator (a module incorporated into the OOMI IDE) and OOMI Translator (a run-time component). But first, the requirements of the Translation Generator and the OOMI Translator are discussed in sections III.B and III.C. The underlying framework (FIOM Framework) implemented in this thesis that both the OOMI IDE and OOMI translator use as the underlying representation of the FIOM is discussed in section III.D.

B. TRANSLATION GENERATOR REQUIREMENTS

The primary objective of the translation generator is to provide maximum assistance to the interoperability engineer. The generator should therefore meet the following requirements:

RG-1: The generator should aim to minimize the common human errors that may occur in the definition of the translation logic.

RG-2: Code Reuse

- The environment should allow the user to add code to perform a specific attribute translation into a central repository, so that the code can be reused in the future for other FCR-CCR translations.
- The environment should allow the modification of code in the code repository to correct errors and allow for requirement changes.
- A major concern in code reuse is whether to reuse through inheritance or through duplication (that is, copying the source code into the new translation) of code fragments.
- A major advantage of inheritance is that improvements to the base code are immediately reflected in all the translations that inherit the base code. On the other hand, this is also a disadvantage when maintaining a larger inheritance base. When the base code is changed, there is a need to perform regression testing on all affected descendents, which may be very costly or even infeasible. Also, changes

to base code may not need to be propagated to all translations that use it. Using inheritance means that the translations are tightly coupled to the central repository.

- Duplication of code has the disadvantage that changes to the base code cannot be easily propagated to the translations that use the code, if propagation is desired. But there is an advantage that changes to the base code is isolated and the user can specify only those translations that need updating and test only the affected translations, rather than having to re-test all translations that use the base code affected. That is, there will be less coupling between translations and the central code repository.

RG-3: Custom code

- The user should have freedom in inserting custom code like Java import statements, additional methods for the translation other than the standard attribute translation methods and additional member variables. Additional member variables are crucial to store temporary computation results that may be required in different stages of translation, providing a more efficient translation process.
- Custom code is required to support situations where the translation is not a simple functional transformation, such as situations requiring access to third party lookup tables (for example, database tables), invocation of other remote objects, etc.

RG-4: Many-to-one attribute translation

- It is possible for the target attribute to be derived from the values of more than one source attribute. In this case, if the target attribute is a mandatory attribute, all source attributes contributing to its derivation are also mandatory in this translation.

RG-5: Preservation of precision in numerical data types

- When translating a numerical attribute to another numerical attribute, care should be taken to ensure there is no loss of precision, this is especially a concern for floating point decimal types. It is proposed that the Java type BigDecimal be used whenever possible, unless there is an overriding concern for performance.

RG-6: Attributes of the target class should not be dependent on each other during the translation.

- That is, in the translation of class A with attributes A1, … , An, from class B, none of Ai should rely on any value Aj ($j \neq i$) . This requirement simplifies the generator's design, since there is no necessity to specify the order in which each target attribute should be translated.

RG-7: The translation generated should be independent of the on-the-wire transmission format, in this case, XML.

The translation generator being implemented for this thesis is a prototype, therefore, it does not fully implement the requirements listed above.

Requirements RG-2 and RG-3 are not addressed in this prototype. It was felt that these two requirements can be adequately addressed with the integration of the OOMI IDE with a commercial Java IDE that will provide management of code libraries and syntax assistance (for example, JBuilder's CodeInsight).

Requirement RG-5 is not implemented, since it is actually a constraint to be placed on the XML data-binding tool.

C. OOMI TRANSLATOR REQUIREMENTS

The main function of the OOMI Translator is to resolve differences in views and representations between a federation's component systems during runtime. The requirements of the OOMI Translator are laid out in this section:

RT-1: Resolve differences in views and representations for the information sent from one system (source system) to another system (destination system).

RT-1.1: The following input to the OOMI Translator are required:

RT-1.1.1: The FIOM for the federation.

RT-1.1.2: An instance of the information of a real-world entity, formatted in the unique representation of the source system in the federation.

RT-1.1.2.1: The information shall be contained in an XML document (source XML document) conforming to the XML schema of the CCR within the FIOM associated with the source system.

RT-1.1.2.2: The uniform resource identifier (URI) identifying the XML schema of the root element of the source XML document is required if the URI cannot be determined from the source XML document. If the URI can be determined from the source XML document, then the URI shall be optional.

RT-1.1.3: The name of the destination system in the federation that will be receiving the information.

RT-1.2: The following output from the OOMI Translator shall be provided:

RT-1.2.1: An instance of the information of the real-world entity, translated from the source system's unique representation and formatted in the unique representation of the destination system.

RT-1.2.1.1: The information shall be contained in an XML document conforming to the XML schema of the CCR within the FIOM associated with the destination system.

RT-1.3: Modeling differences between source and destination systems shall be automatically resolved upon receiving an XML document from the source system.

RT-2: Communications.

RT-2.1: Manage connection with the source system to receive the information sent from the source system.

RT-2.1.1: During communication with the source system, if the URI cannot be determined from the source XML document, then the uniform resource identifier (URI) identifying the XML schema of the root element of the source XML document is required from the source system. If the URI can be determined from the source XML document, then the source system does not necessarily have to send the URI.

RT-2.2: Manage connection with the destination system to send the translated information to the destination system.

The OOMI Translator prototype implemented in this thesis does not address requirement *RT-2*.

D. THE FIOM FRAMEWORK

The FIOM Framework, defined in the package *mil.navy.nps.cs.oomi.fiom*, provides the functionalities required for implementing the OOMI IDE and OOMI Translator. The set of Java interfaces in this framework defines the objects modeling the FIOM. The set of Java classes defined in the *mil.navy.nps.cs.oomi.impl* package implements these interfaces and provides persistent storage to an XML file.

1. FIOM Framework Requirements

The requirements of the FIOM Framework are as follows:

RF-1: Provide a Java object model implementing the FIOM.

RF-2: Provide application programming interfaces (APIs) to manage a collection of FIOMs.

RF-3: Provide APIs to store the collection of FIOMs to persistent storage and to restore the collection from persistent storage.

2. FIOM Framework Constraints

The following constraints were imposed on the FIOM Framework:

CF-1: Uniqueness of names within a FIOM.

CF-1.1: Within a FIOM, each component system is identified by a name.

CF-1.2: Within a FIOM, each FE is uniquely identified by a name.

CF-1.3: Identifiers of a CCR.

CF-1.3.1: Within a FIOM, each CCR is uniquely identified by the pair of values comprising the name of its component system and the name of the CCR.

CF-1.3.2: Within a FIOM, each CCR is uniquely identified by the pair of values comprising the name of its component system and the uniform resource identifier (URI) of the XML schema that this CCR corresponds to.

CF-2: Uniqueness of names within an FE.

CF-2.1: Within an FE, each FEV is uniquely identified by a name.

CF-3: Each FEV is exactly represented by one and only one FCR.

3. The FIOM Framework Components

Within the constraints described in section III.D.2, the FIOM Framework was developed to satisfy the requiremenrts for the OOMI IDE and the OOMI Translator, outlined in sections III.B and III.C, respectively. Figure III-2 provides an overview of the FIOM Framework.

The FIOM Framework consists of components from the mil.navy.nps.cs.oomi, mil.navy.nps.cs.oomi.impl and mil.navy.nps.cs.oomi.fiom packages. The mil.navy.nps.cs.oomi and mil.navy.nps.cs.oomi.fiom packages were implemented using pure interfaces so that future improvements to the underlying implementation of the OOMI IDE and OOMI Translator can performed easily by replacing only the classes that implement these interfaces. In this implementation, the classes that implement these interfaces are coded in the mil.navy.nps.cs.oomi.impl package. The complete source code listing of the FIOM Framework is included in Appendix A.

The FIOM Framework was designed to encapsulate the underlying details of persistent storage. Developers requiring access to the FIOMs in the OOMI database do not need to deal with the details of access to persistent storage. The classes used to implement the persistent storage of a collection of FIOMs in the form of an XML file are also contained in the mil.navy.nps.cs.oomi.impl package.

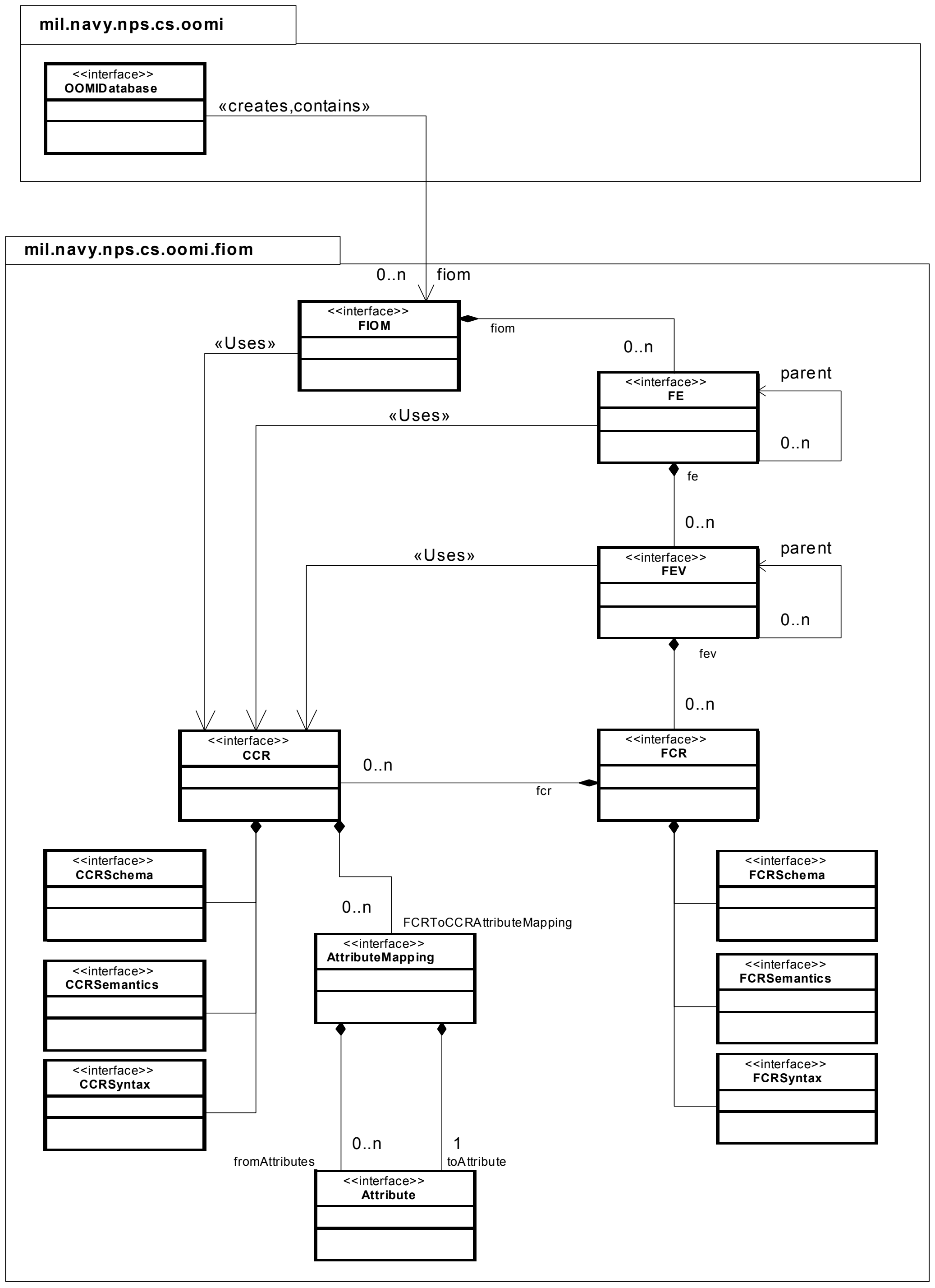


Figure III-2. mil.navy.nps.cs.oomi and mil.navy.nps.cs.oomi.fiom packages.

The *OOMIDatabase* interface in the package mil.navy.nps.cs.oomi is to be used by the OOMI IDE to handle a collection of FIOMs. It is also meant to be used by the OOMI Translator to load a specific *FIOM* from persistent storage, for use by the *Translator* object.

Finally, the components that define the FIOM and methods to manipulate those components are contained in the package mil.navy.nps.cs.oomi.fiom. The first component, the *FIOM* interface is to be used by *Translator* object to perform translations.

The *FCR* interface provides the meta information of the FCR, composed of *FCRSchema*, *FCRSemantics* and *FCRSyntax*. *FCRSemantics* and *FCRSyntax* are currently not required and thus not implemented. They are meant to be used by the OOMI IDE correlator module. *FCRSchema* interface contains the complete information on the attributes of the FCR and the name of the Java class (descendent of *FCRInstance*) which when instantiated, will hold the values of the FCR's attributes.

The *CCR* interface provides the meta information of the CCR, composed of *CCRSchema*, *CCRSemantics* and *CCRSyntax*. *CCRSemantics* and *CCRSyntax* are currently not required and thus not implemented. They are meant to be used by the OOMI IDE correlator module. *CCRSchema* interface contains the complete information on the attributes of the CCR and the name of the Java class (descendent of *CCRInstance*) which when instantiated, will hold the values of the CCR's attributes. As shown in Figure III-3, the Castor data-binding tool is used in the OOMI IDE to generate an object representation of the component system's XML schema. The generated class shall be a descendent of *CCRInstance*. A CCR is uniquely defined by the *SystemName* and *CCRName* pair and also uniquely identified by the *SystemName* and *XMLNameSpaceURI* pair within the FIOM. The *CCR* interface also provides access to information on whether an attribute of its FCR is mandatory, based on the FCRtoCCR attribute mappings contained in the object implementing this interface.

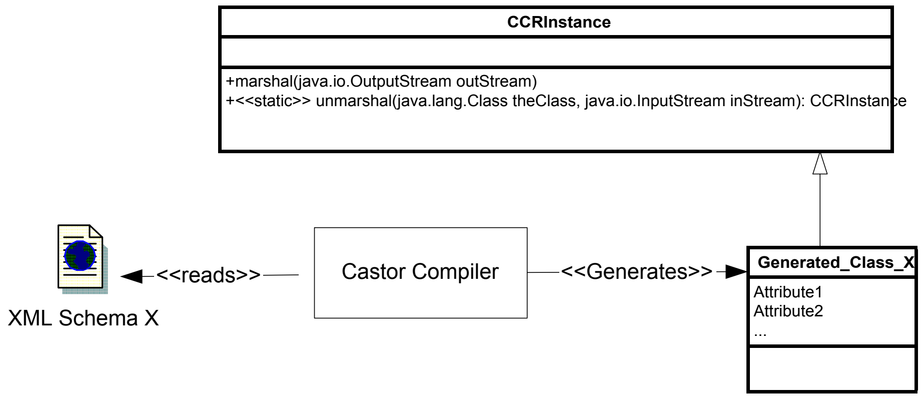

CCRInstance
+marshal(java.io.OutputStream outStream)
+<<static>> unmarshal(java.lang.Class theClass, java.io.InputStream inStream): CCRInstance
XML Schema X
<<reads>>
Castor Compiler
<<Generates>>
Generated_Class_X
Attribute1
Attribute2
...

Figure III-3. Data-Binding with Castor.

The *Attribute* interface component describes the attribute of either an FCR or CCR, and contains information like the name, data type and the minimum and maximum number of elements allowed if the attribute is an array.

Finally, the *AttributeMapping* interface describes the mapping from an array of attributes to a specific attribute. It is used to hold the definition of attribute mappings between an FCR and CCR.

E. SUMMARY

The requirements of the Translation Generator and the OOMI Translator form the basis for the development of the prototypes. The FIOM Framework defines the object model implementing the FIOM and APIs for manipulating and managing persistence of the model.

The Translation Generator and the OOMI Translator prototypes developed for this thesis are discussed in detail in chapters IV and V, respectively.

IV. TRANSLATION GENERATOR

A. INTRODUCTION

A Federation Interoperability Object Model (FIOM) is created for a system federation to capture the information and operations shared between different systems. The OOMI translator performs the conversion between models of the real-world entities whose information and operations are shared among systems using translations contained in the FIOM. An automatic translation generator is provided as part of the OOMI IDE to provide computer-aided assistance to the Interoperability Engineer (IE) in defining the translations required.

FIOM translations are used to resolve representational differences among the attributes and operation signatures used to define a component model of a real-world entity. The process of defining the transformations between attribute and operation signatures is potentially error-prone, therefore the OOMI IDE should aim to automate the processes of translation creation in order to minimize chances of human error wherever possible. The translation generation algorithm discussed in this chapter provides such automation to translation construction.

For the OOMI IDE prototype, the CCRs and FCRs are represented internally as Java classes (CCRInstance class and FCRInstance class, respectively) produced through a data-binding process, using the CASTOR data-binding tool. The translation generator will therefore work directly with Java classes and utilize the Java reflection API to work on compiled Java classes.

B. DEFINING ATTRIBUTE TRANSLATIONS

This section will describe the process of how the (IE) uses the OOMI IDE translation generator module to create the framework for a translation definition. This translation "skeleton" will be completed by the IE using both newly created and previously saved transformations. A simple graphical user interface was developed for the generator.

As discussed in section II.B.2, the OOMI uses an intermediate representation during the translation process in order to solve the $O(n^2)$ translations problem. The process thus requires definition of two translations for each source-destination system pair. The first translation is used to convert between the source system Component Class Representation (CCR) and the intermediate Federation Class Representation (FCR). The second translation is used to convert from the intermediate FCR to the destination CCR. Thus the translation generation module is used to create translations between FCRs and CCRs.

Figure IV-1 shows the user interface used by the translation generation module to create such FCR-CCR translations. As seen in the figure, section A displays an FCR and its attributes, while section B shows a CCR with its attributes. Composite FCR or CCR components are displayed through expansion of the FCR or CCR tree structure.

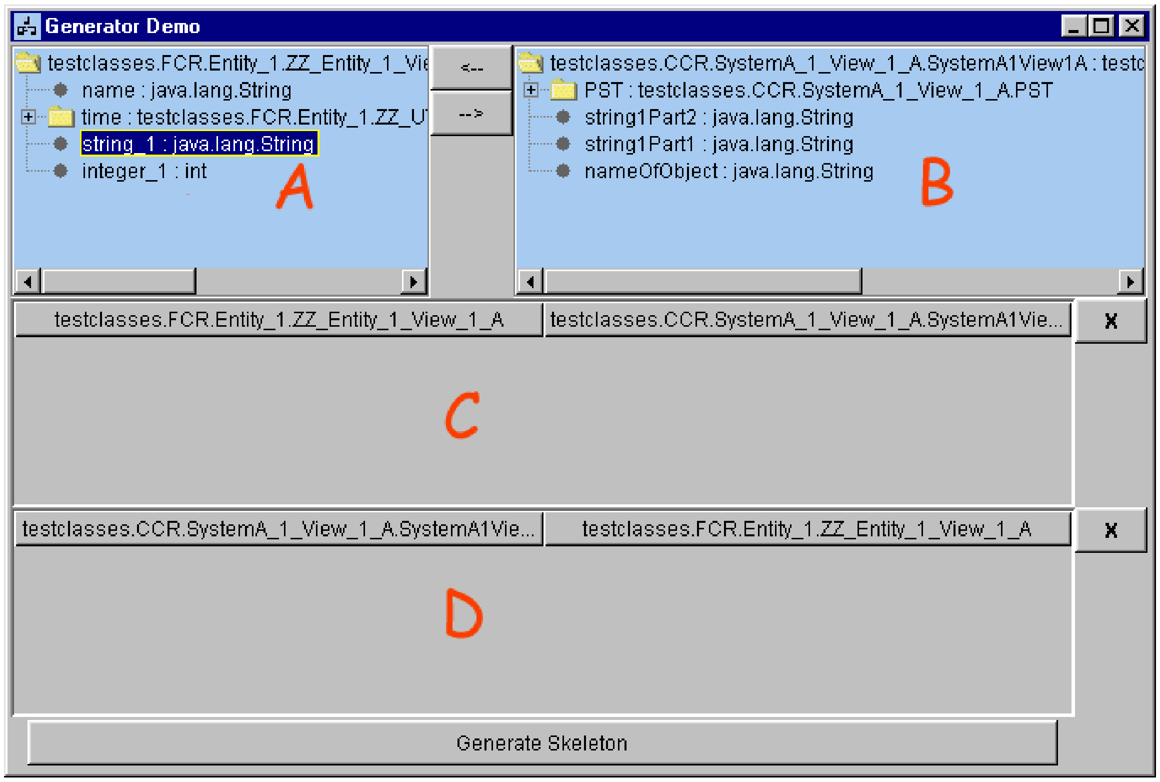
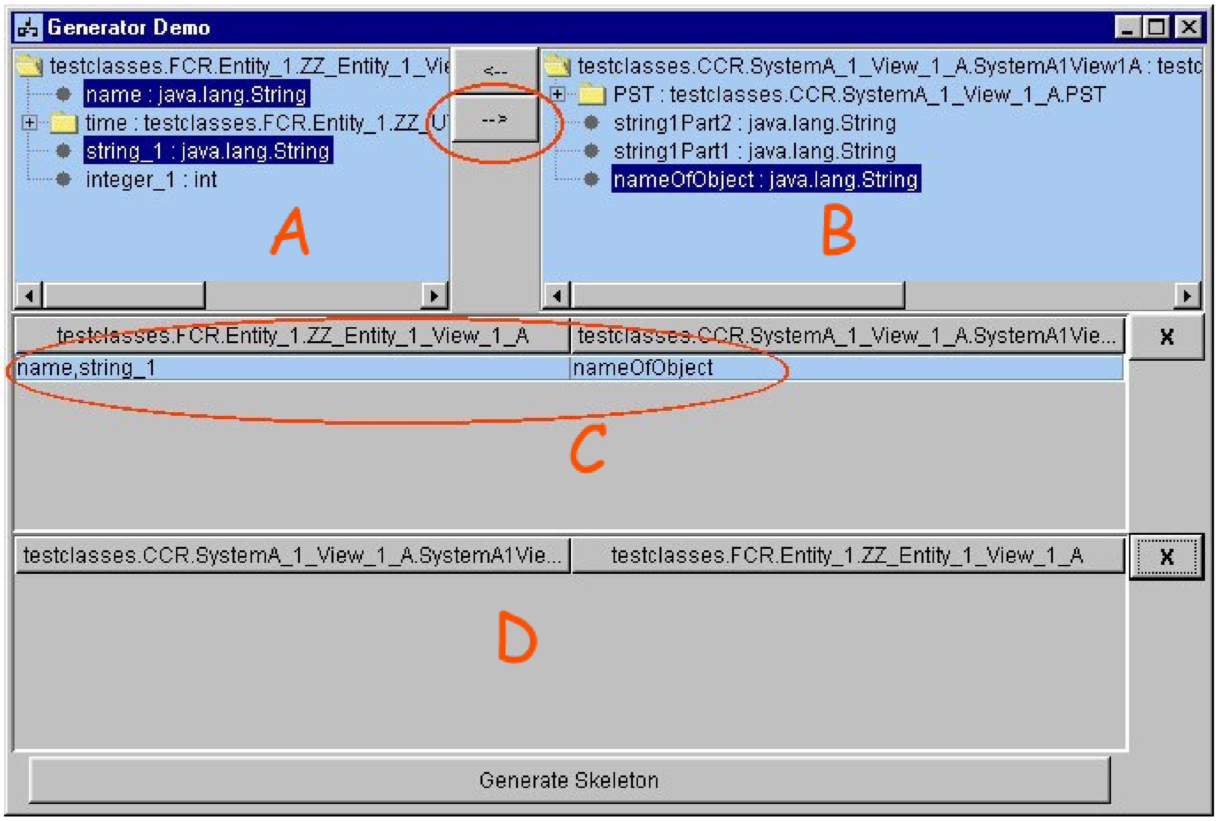

Generator Demo
testclasses.FCR.Entity_1.ZZ_Entity_1_Vie
name : java.lang.String
time : testclasses.FCR.Entity_1.ZZ_U
string_1 : java.lang.String
integer_1 : int
A
<--
-->
testclasses.CCR.SystemA_1_View_1_A.SystemA1View1A : testc
PST : testclasses.CCR.SystemA_1_View_1_A.PST
string1Part2 : java.lang.String
string1Part1 : java.lang.String
nameOfObject : java.lang.String
B
testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_A
testclasses.CCR.SystemA_1_View_1_A.SystemA1Vie...
x
C
testclasses.CCR.SystemA_1_View_1_A.SystemA1Vie...
testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_A
x
D
Generate Skeleton

Figure IV-1. The Translation Generator module.

To define a mapping between the FCR attributes name and string_1 and the CCR attribute nameOfObject, the IE selects these attributes as shown and hits the "→" button, creating a mapping from "name,string_1" to "nameOfObject"; the FCR to CCR mapping is displayed in section C:

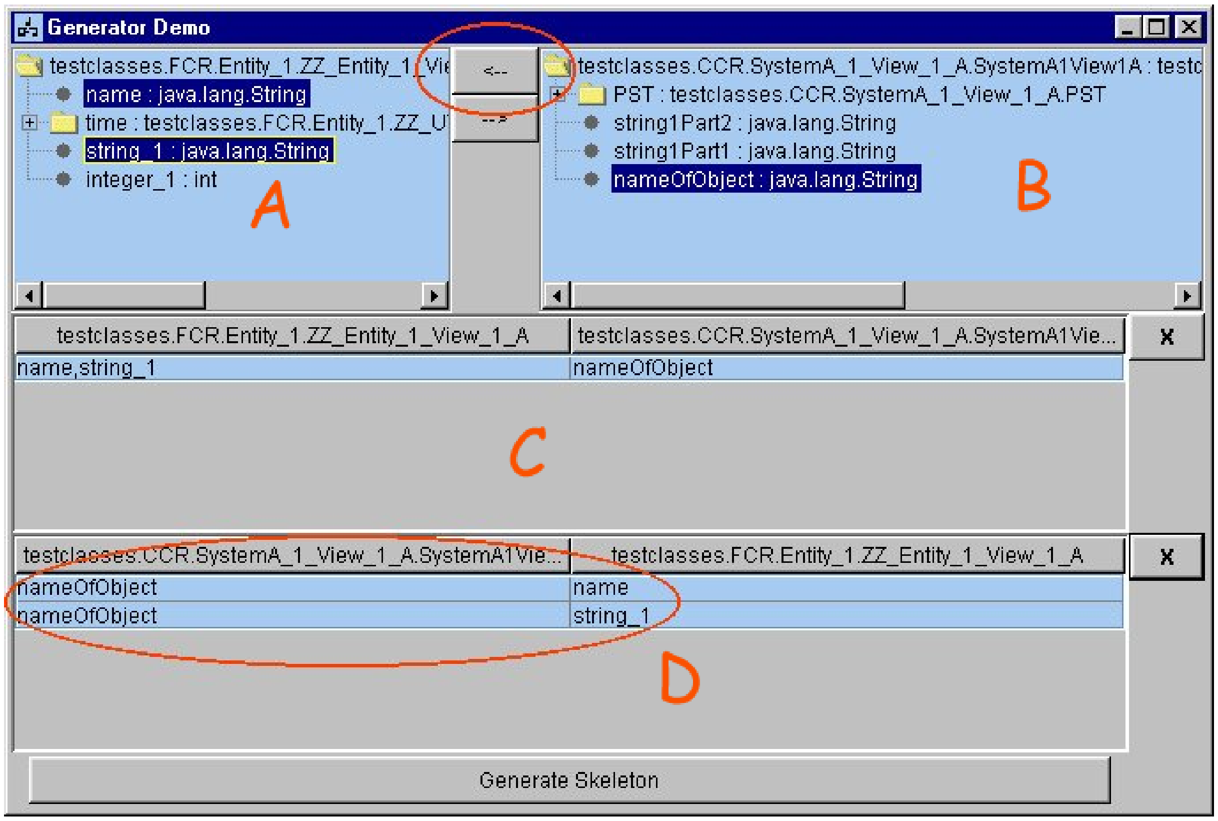

Generator Demo
testclasses.FCR.Entity_1.ZZ_Entity_1_Vie
name : java.lang.String
time : testclasses.FCR.Entity_1.ZZ_U
string_1 : java.lang.String
integer_1 : int
<--
-->
testclasses.CCR.SystemA_1_View_1_A.SystemA1View1A : testc
PST : testclasses.CCR.SystemA_1_View_1_A.PST
string1Part2 : java.lang.String
string1Part1 : java.lang.String
nameOfObject : java.lang.String
A
B
C
D
X
Generate Skeleton

testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_A	testclasses.CCR.SystemA_1_View_1_A.SystemA1Vie...
name,string_1	nameOfObject

testclasses.CCR.SystemA_1_View_1_A.SystemA1Vie...	testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_A

Figure IV-2. Defining FCRInstance to CCRInstance attribute mapping.

To define the reverse mapping, the IE uses the "←" button, and the CCR to FCR mapping is displayed in section D:



testclasses.FCR.Entity_1.ZZ_Entity_1_Vi
name : java.lang.String
time : testclasses.FCR.Entity_1.ZZ_U
string_1 : java.lang.String
integer_1 : int

testclasses.CCR.SystemA_1_View_1_A.SystemA1View1A : testc
PST : testclasses.CCR.SystemA_1_View_1_A.PST
string1Part2 : java.lang.String
string1Part1 : java.lang.String
nameOfObject : java.lang.String

testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_A	testclasses.CCR.SystemA_1_View_1_A.SystemA1Vie...
name,string_1	nameOfObject

testclasses.CCR.SystemA_1_View_1_A.SystemA1Vie...	testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_A
nameOfObject	name
nameOfObject	string_1

Figure IV-3. Defining CCRInstance to FCRInstance attribute mapping.

Finally, the IE clicks the "Generate Skeleton" button to generate the translation code skeleton. The OOMI IDE then provides the capability to modify the translation code skeleton to add functional or other transformations as necessary to resolve representational differences.

C. TRANSLATION GENERATION PROCESS

The OOMI IDE creates a *GeneratorUI.Plugin* object which implements the user interface shown in Figure IV-1. The translation generation process begins with the definition of attribute translations, described in section IV.B, and is completed when the IE clicks on the "Generate Skeleton" button. The button click invokes the *generateSkeleton(...)* method of *GeneratorUI.Plugin* object, *plugin*, to generate the code skeleton. This process is illustrated in Figure IV-4.

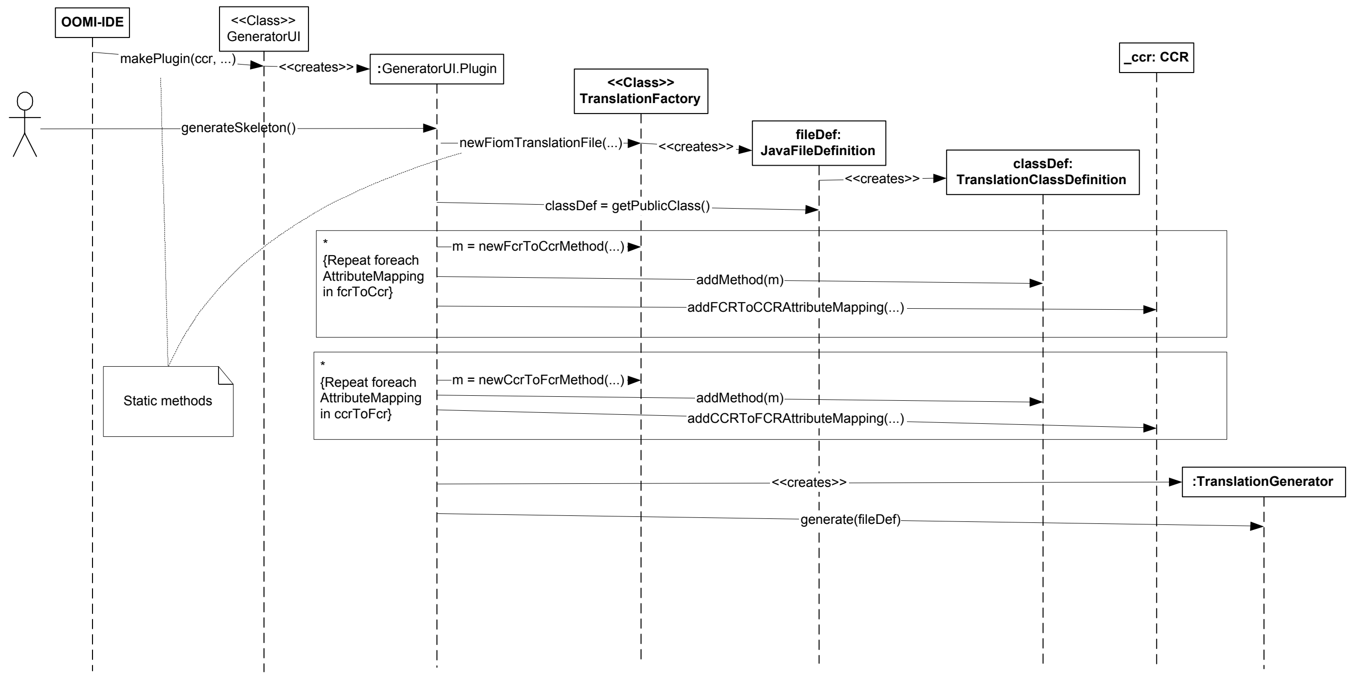

OOMI-IDE
<<Class>> GeneratorUI
:GeneratorUI.Plugin
<<Class>> TranslationFactory
fileDef: JavaFileDefinition
classDef: TranslationClassDefinition
_ccr: CCR
:TranslationGenerator
makePlugin(ccr, ...)
<<creates>>
generateSkeleton()
newFiomTranslationFile(...)
<<creates>>
<<creates>>
classDef = getPublicClass()
* {Repeat foreach AttributeMapping in fcrToCcr}
m = newFcrToCcrMethod(...)
addMethod(m)
addFCRToCCRAttributeMapping(...)
* {Repeat foreach AttributeMapping in ccrToFcr}
m = newCcrToFcrMethod(...)
addMethod(m)
addCCRToFCRAttributeMapping(...)
Static methods
<<creates>>
generate(fileDef)

Figure IV-4. Sequence diagram for the translation generation process.

As shown in Figure IV-4, the *generateSkeleton(...)* method performs the following:

1. Create a *JavaFileDefinition* object, *fileDef*, by invoking the static method *newFiomTranslationFile()* of *TranslationFactory* class. The object, *fileDef*, defines the content of the Java source file that contains the code skeleton to be generated.

2. Invokes *getPublicClass()* method of *fileDef* to retrieve the *TranslationClassDefinition* instance, *classDef*, from *fileDef*. The *TranslationClassDefinition* instance defines the methods of the Java class implementing the translation.

3. For each *AttributeMapping* in the *TranslationMap*, *plugin.fcrToCcr*,
   add this attribute mapping to the *CCR*, by calling
   addFCRToCCRAttributeMapping(...) of the *CCR*,
   call *TranslationFactory.newFcrToCcrMethod(...)* to create an
   AttributeTranslationMethod object, and add this object to *classDef*, by invoking
   classDef.addMethod(...)

4. For each *AttributeMapping, attr,* in the *TranslationMap*, *ccrToFcr*,
   call *TranslationFactory.newCcrToFcrMethod(...)* to create a
   AttributeTranslationMethod object, and add this object to *classDef*, by invoking
   classDef.addMethod(...)

5. Generate the source code for the abstract methods *toFCR* and *toCCR*, defined in *AbstractTranslation*

6. Create a new *TranslationGenerator* object and invoke its *generate(...)* method to generate the source code for the skeleton.

D. TRANSLATION GENERATOR IMPLEMENTATION

The classes used to implement the Translation Generation process discussed in section IV.C are shown in Figure IV-5. A brief description of each of these component classes follows the figure. The complete source code listing of the Translation Generator is included in Appendix B.

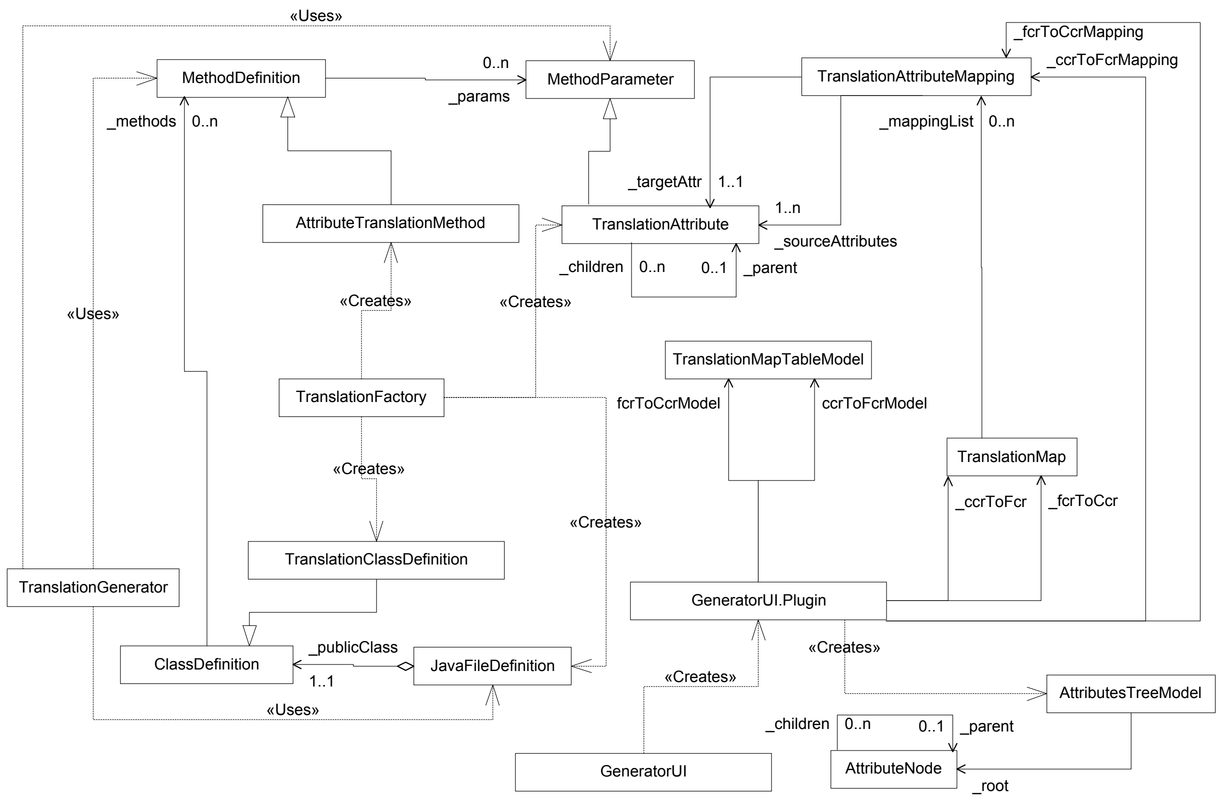


Figure IV-5. Class diagram of the Translation Generator.

GeneratorUI – A factory class that will create an instance of the inner class *GeneratorUI.Plugin*, which implements the user-interface for the Translator Generator module.

GeneratorUI.Plugin – The inner class implementing the user-interface for the Translator Generator plugin to the OOMI IDE.

AttributesTreeModel and *AttributeNode* – Provides the data model required by the Java Swing JTree components (sections A and B in Figure IV-1) for the display of a class and its attributes.

TranslationMapTableModel – Provides the data model required by the Java Swing JTable components (sections C and D in Figure IV-1) for the display of attribute mappings.

TranslationFactory – A factory to create various FIOM translation specific class instances, suhc as instances of *AttributeTranslationMethod* and *JavaFileDefinition*.

JavaFileDefinition – Defines a Java source file containing the code skeleton to be generated. It contains an instance of *ClassDefinition* representing the public class contained in the Java source file.

ClassDefinition – An instance of this class defines a Java class that implements the translation being defined. The set of *MethodDefinition* objects contained within a *ClassDefinition* object represents the methods of the class defined by *ClassDefinition*

TranslationClassDefinition – A subclass of *ClassDefinition* specialized to represent the source code of a subclass of *AbstractTranslation*. *AbstractTranslation* class, described in detail in section III.E, is used by the OOMI Translator to represent a translation.

MethodDefinition – An instance of this class represents the definition of a Java method.

AttributeTranslationMethod – A subclass of *MethodDefinition* defining a FIOM attribute translation. Contains additional information required to generate translations, specifying the target attribute that this method is meant to translate to.

MethodParameter – An instance of this class represents a parameter of a method defined by *MethodDefinition*.

TranslationAttribute –Describes an attribute of a class in the FIOM. An attribute, if not a Java primitive type, can be a parent to other attributes, forming a tree structure that reflects the structure of this non-primitive attribute.

TranslationMap – Keeps a collection of attribute mappings between two classes. The collection will be composed of one or more *TranslationAttributeMapping* objects.

TranslationAttributeMapping – Holds the many-to-one mapping from attributes of one class to the attribute of another class.

TranslationGenerator – Contains the logic to generate Java source code for a translation, based on the information contained in a *JavaFileDefinition* object.

E. GENERATED TRANSLATION CLASS HIERARCHY

All translations generated are sub-types of the class *AbstractTranslation* in the package *mil.navy.nps.cs.oomi.fiom*, depicted in Figure IV-6. The OOMI translator makes use of the abstract base class *AbstractTranslation* to carry out the translations. Specifically, the OOMI translator makes use of the following abstract methods defined in the abstract base class:

- *toCCR(fcr: FCRInstance)* – translates the specified FCRInstance object to a CCRInstance object
- *toFCR(ccr: CCRInstance)* – translates the specified CCRInstance object to an FCRInstance object

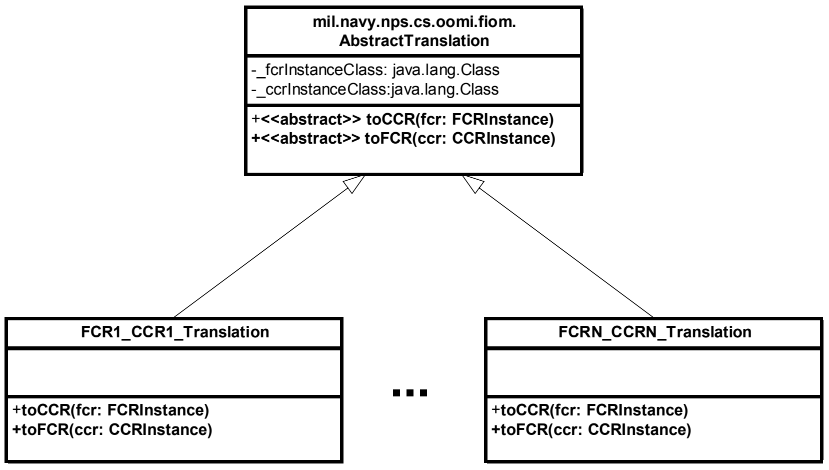

mil.navy.nps.cs.oomi.fiom.
AbstractTranslation
-_fcrInstanceClass: java.lang.Class
-_ccrInstanceClass:java.lang.Class
+<<abstract>> toCCR(fcr: FCRInstance)
+<<abstract>> toFCR(ccr: CCRInstance)
FCR1_CCR1_Translation
+toCCR(fcr: FCRInstance)
+toFCR(ccr: CCRInstance)
...
FCRN_CCRN_Translation
+toCCR(fcr: FCRInstance)
+toFCR(ccr: CCRInstance)

Figure IV-6. Class hierarchy of the generated translations.

The IE will use the Translation Generator to create an FCR_CCR_Translation class that extends the *AbstractTranslation* class for each FCR-CCR pair defined in the FIOM as indicated in Figure IV-6. Each FCR_CCR_Translation class will implement the *toCCR* and *toFCR* abstract methods from *AbstractTranslation*. These methods will invoke a number of sub-methods to perform the translation on an attribute-by-attribute basis as follows:

- method *toCCR* will invoke one or more methods with names having the prefix fcrToCcr_; these methods are responsible for deriving one CCR attribute from one or more FCR attributes
- method *toFCR* will invoke one or more methods with names having the prefix ccrToFcr_; these methods are responsible for deriving one FCR attribute from one or more CCR attributes

Two method signatures are possible for translating attributes { B_1 , B_2 , … , B_n } of a certain FCR to attribute ccrA of a corresponding CCR:

1. public int fcrToCcr_ccrA(FCRInstance fcr)
   - This does not restrict the attributes that the method can access in the translation, making the optional/mandatory aspects hard to handle
   - The fields in the parameter *fcr* can be inadvertently modified by the code in the method, this is a potential source of errors
2. public int fcrToCcr_ccrA(B_1 , B_2 , … , B_n)
   - This restricts the method body on the source attributes that it can access, thereby providing an easy way for the tool to determine if all required attributes B_1 , B_2 , … , B_n are present and for the tool to generate the dependency of any attribute
   - This format also provides a contract on what attributes can be used in the computation of attribute ccrA
   - The fields of the source object cannot be modified within the method, as long as all the parameters B_1 , B_2 , … , B_n are immutable objects or primitive data types

Option 2 was determined to be more helpful to the interoperability engineer.

The ccrToFcr_ methods are symmetrical to the above:

- public int ccrToFcr_fcrB(CCRInstance ccr)
- public int ccrToFcr_fcrB(A_1 , A_2 , … , A_n)

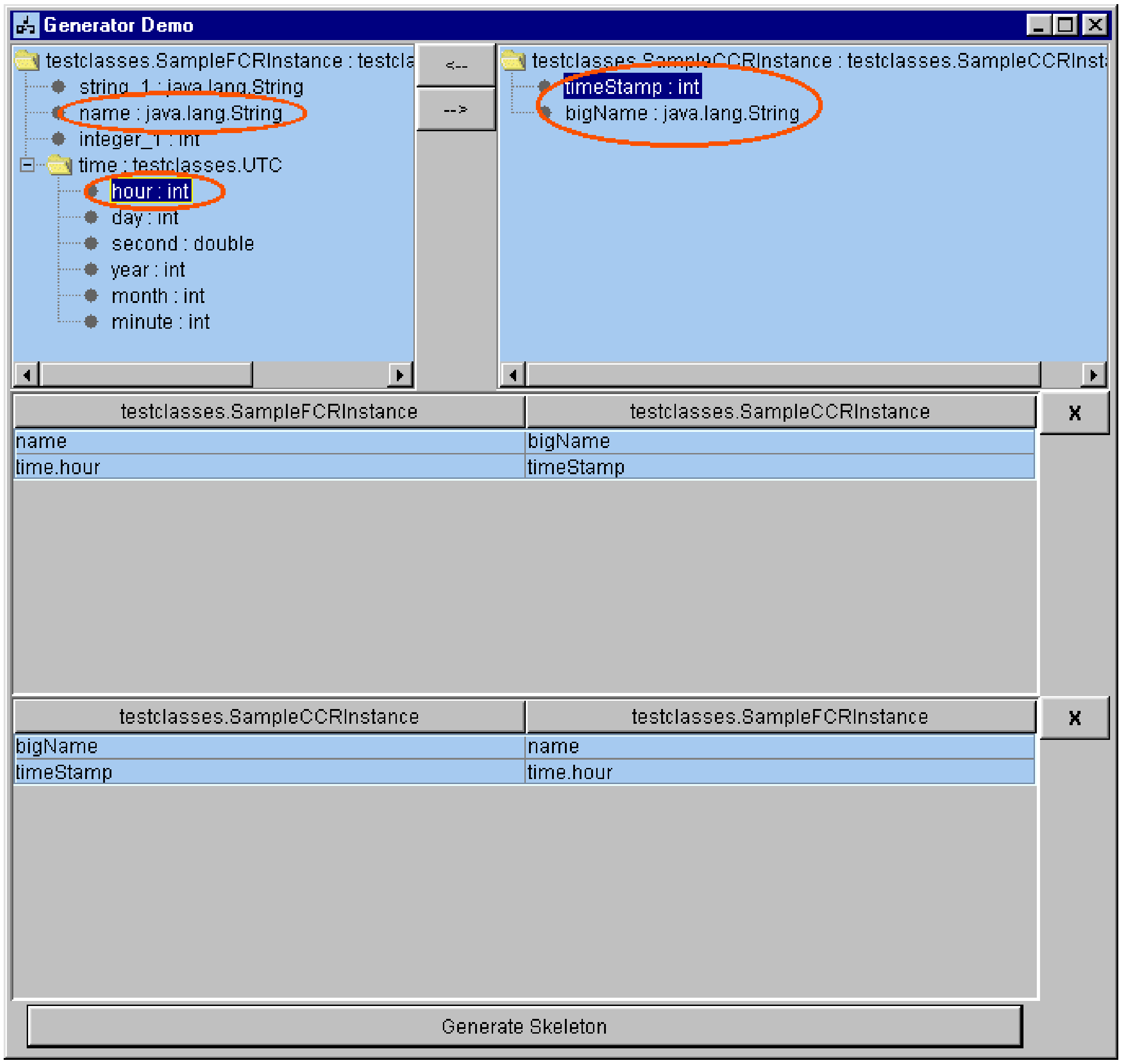
F. THE GENERATED TRANSLATION SKELETON

The translation skeleton generated by the generator in Figure IV-7 below is an example of what the code the Translation Generator will provide for an FCR-CCR pair based on the attribute mappings specified by the IE.

```
package Translations.testclasses.testclasses;

//%% begin import
// -- Enter your additional imports here.
//%% end import

public class SampleFCRInstance__SampleCCRInstance extends
       mil.navy.nps.cs.oomi.fiom.AbstractTranslation {
  //%% begin fields
  // -- Enter your custom fields here.
  //%% end fields

  public  SampleFCRInstance__SampleCCRInstance (  ) {
    //%% begin method_body -- Enter method body below:
    init( testclasses.SampleFCRInstance.class, testclasses.SampleCCRInstance.class );
    //%% end method_body
  }
  public java.lang.String fcrToCcr_bigName ( java.lang.String name ) {
    //%% begin method_body -- Enter method body below:
    //%% end method_body
  }
  public int fcrToCcr_timeStamp ( int hour ) {
    //%% begin method_body -- Enter method body below:
    //%% end method_body
  }
  public java.lang.String ccrToFcr_name ( java.lang.String bigName ) {
    //%% begin method_body -- Enter method body below:
    //%% end method_body
  }
  public int ccrToFcr_time_hour ( int timeStamp ) {
    //%% begin method_body -- Enter method body below:
    //%% end method_body
  }
  public mil.navy.nps.cs.oomi.fiom.CCRInstance toCCR (
       mil.navy.nps.cs.oomi.fiom.FCRInstance fcrInstance ) {
    //%% begin method_body -- Enter method body below:
    testclasses.SampleFCRInstance obj = (testclasses.SampleFCRInstance) fcrInstance;
    testclasses.SampleCCRInstance result = new testclasses.SampleCCRInstance();

    result.setBigName(fcrToCcr_bigName(obj.getName()));
    result.setTimeStamp(fcrToCcr_timeStamp(obj.getTime().getHour()));

    return result;
    //%% end method_body
  }
  public mil.navy.nps.cs.oomi.fiom.FCRInstance toFCR (
       mil.navy.nps.cs.oomi.fiom.CCRInstance ccrInstance ) {
    //%% begin method_body -- Enter method body below:
    testclasses.SampleCCRInstance obj = (testclasses.SampleCCRInstance) ccrInstance;
    testclasses.SampleFCRInstance result = new testclasses.SampleFCRInstance();

    result.setTime(new testclasses.UTC());

    result.setName(ccrToFcr_name(obj.getBigName()));
    result.getTime().setHour(ccrToFcr_time_hour(obj.getTimeStamp()));

    return result;
    //%% end method_body
  }
  //%% begin methods
  // -- Enter your custom methods here.
  //%% end methods
}
```

Figure IV-7.   Sample Translation Skeleton.

The sample code illustrates the translation defined for the classes *SampleFCRInstance* and *SampleCCRInstance*, shown in Figure IV-8. The translations between the *SampleFCRInstance* attributes "Name" and "Hour" (Hour is an attribute of the attribute Time in *SampleFCRInstance*), and the *SampleCCRInstance* attributes "BigName" and "TimeStamp" are shown.



testclasses.SampleFCRInstance	testclasses.SampleCCRInstance
name	bigName
time.hour	timeStamp

testclasses.SampleCCRInstance	testclasses.SampleFCRInstance
bigName	name
timeStamp	time.hour

Figure IV-8. Defining mappings between *SampleFCRInstance* and *SampleCCRInstance*.

The code contains the following methods of interest:

- *public java.lang.String fcrToCcr_bigName (java.lang.String name)* – given the value of the FCRInstance's name attribute, translates it to the CCRInstance's BigName attribute and returns the result
- *public int fcrToCcr_timeStamp (int hour)* – given the value of the FCRInstance's hour attribute, translates it to the CCRInstance's TimeStamp attribute and returns the result
- *public java.lang.String ccrToFcr_name (java.lang.String bigName)* – given the value of the CCRInstance's BigName attribute, translates it to the FCRInstance's Name attribute and returns the result
- *public int ccrToFcr_time_hour (int timeStamp)* – given the value of the CCRInstance's TimeStamp attribute, translates it to the FCRInstance's Hour attribute and returns the result. This method is named *ccrToFcr_time_hour* since the "Hour" attribute is actually a sub-attribute in the "Time" attribute of *SampleFCRInstance*
- *toCCR* – calls each of the *fcrToCcr_* methods to convert the specified *FCRInstance* into a *CCRInstance*
- *toFCR* – calls each of the *ccrToFcr_* methods to convert the specified *CCRInstance* into an F*CRInstance*

Special markers "`//%%`" are used to delimit the sections of code that are generated by the generator and sections where the interoperability engineer can directly manipulate the code. This is meant for future implementation of round-trip engineering capability, where the translations can be directly imported back into the IDE for editing and re-generation without overwriting the code entered by the interoperability engineer. The IE will enter the code for performing the attribute translation between the lines

```
//%% begin method_body -- Enter method body below:
```

and

```
//%% end method_body
```

for all methods with names beginning with `ccrToFcr_` or `fcrToCcr_`.

G. SUMMARY

The Translation Generator described in this chapter provides a simple graphical user interface for defining mappings between FCR attributes and CCR attributes, and the capability to generate the Java source code for a skeleton translation class that will perform the translations between an F*CRInstance* object and a *CCRInstance* object.

The skeleton translation classes generated by the Translation Generator will be used by the OOMI Translator (described in chapter V) to resolve differences in views and representations between the source and destination systems.

THIS PAGE INTENTIONALLY LEFT BLANK

V. OOMI TRANSLATOR

A. INTRODUCTION

The OOMI Translator is the run-time component responsible for resolving heterogeneities among systems to enable interoperation. The Translator converts information shared between two systems from its source system model to the model expected by the destination system. In its current implementation, information exported from or imported to a component system is provided in the form of an XML document. This XML document conforms to an XML schema that defines a real-world entity whose attributes and operations are being shared between systems.

The current implementation of the Translator translates source XML documents into destination XML documents by utilizing the translations defined in the FIOM.

The Translator is implemented as a Java class that is independent of the transport mechanism to be used to perform the communication between systems. The Translator class is designed to be easily incorporated into any application that will utilize the OOMI for interoperability.

Figure V-1 provides an overview of the Translator's function and depicts its relationship with the other components of the OOMI. In particular, Figure V-1 depicts how the OOMI Translator uses translations and component model relationships contained in the OOMI IDE developed FIOM to perform the translation between source and destination model XML documents.

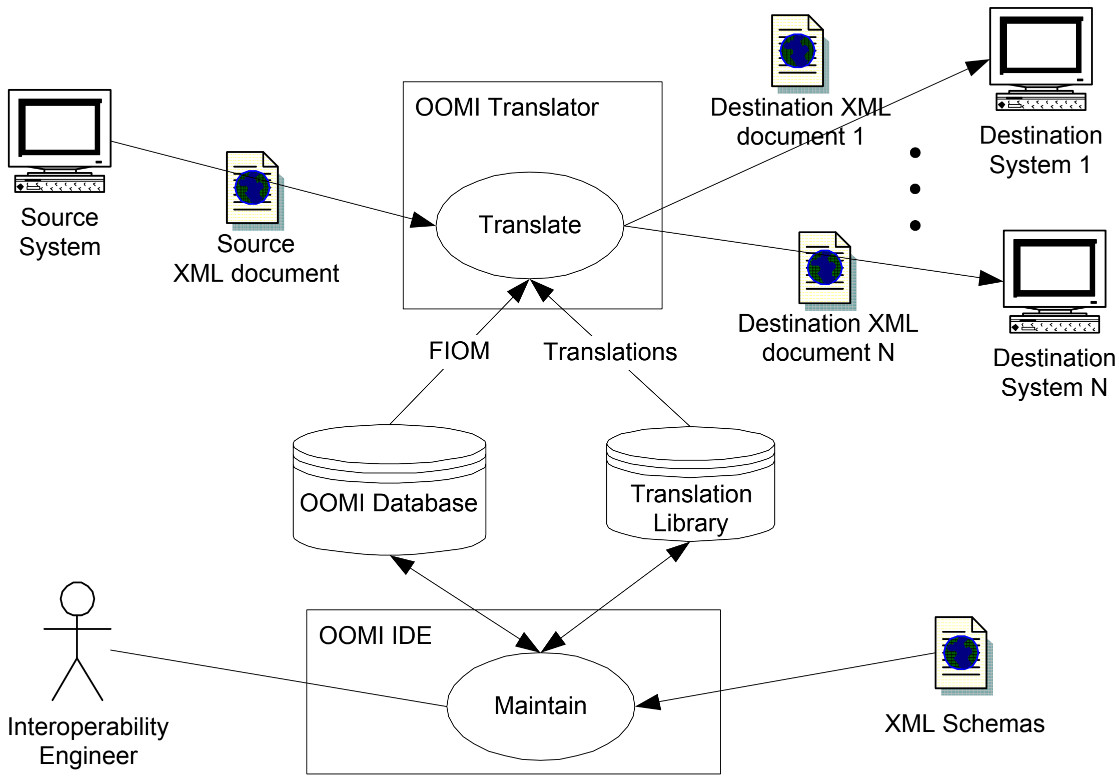


Figure V-1. Overview of the Translator and the IDE.

Figure V-2 illustrates the typical scenarios used for converting a source system XML document to a model usable by a destination system. A top-level view of that process is provided in the discussion following the figure.

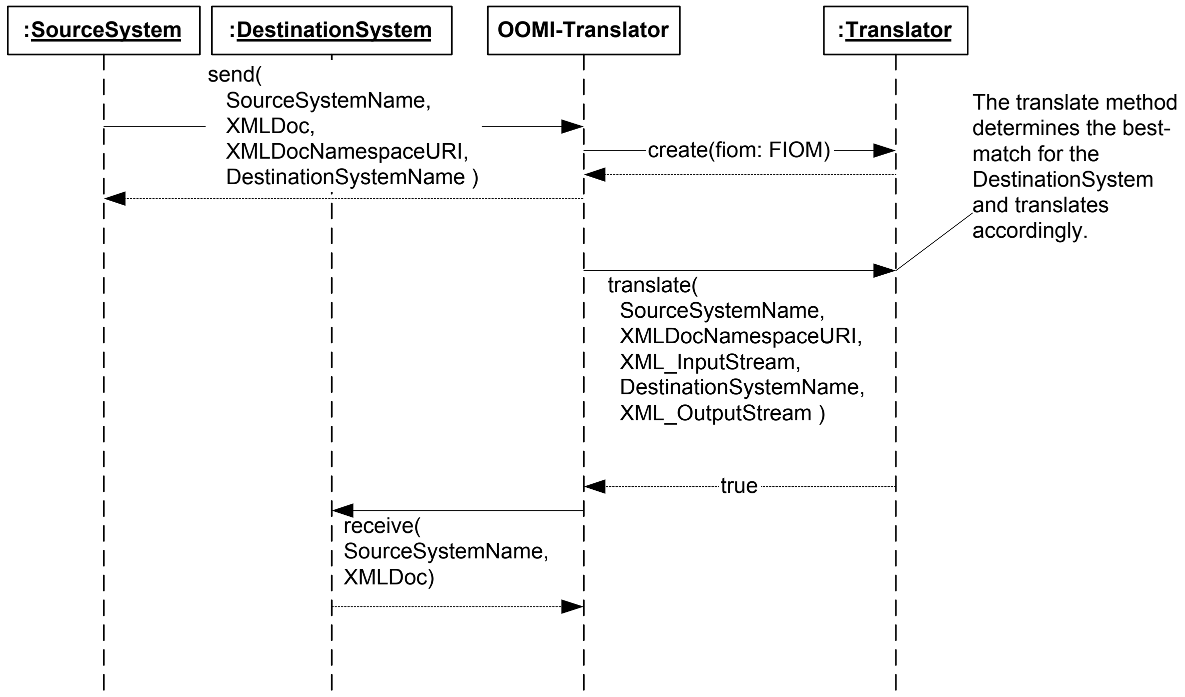

:SourceSystem
:DestinationSystem
OOMI-Translator
:Translator
send(SourceSystemName, XMLDoc, XMLDocNamespaceURI, DestinationSystemName)
create(fiom: FIOM)
The translate method determines the best-match for the DestinationSystem and translates accordingly.
translate(SourceSystemName, XMLDocNamespaceURI, XML_InputStream, DestinationSystemName, XML_OutputStream)
true
receive(SourceSystemName, XMLDoc)

Figure V-2. Typical scenario of a translation.

send(...) – In the first step, the source system packages the information it needs to export into an XML document (XMLDoc), establishes a connection to the OOMI Translator and sends XMLDoc to it. Additional information required by the OOMI Translator includes the source system name, the URI of the XML schema defining XMLDoc and the destination system's name. The URI of the XML schema of XMLDoc could be extracted from XMLDoc itself by the OOMI Translator if it is included in XMLDoc (inclusion of the reference to the URI of an XML document's schema is optional), with a slight overhead on the part of the OOMI Translator.

create(...) – Next, the OOMI Translator creates an instance of the Translator class, with a FIOM instance defined for this federation.

translate(...) – The OOMI Translator then initiates the translation process by invoking the *translate()* method of the translator, specifying the name of the source system, the XML namespace URI of the schema for XMLDoc, an input stream instance supplying the content of the XML document, the name of the destination system and an output stream instance where the translated document can be written to. The *translate(...)* method will determine the best translation for the destination system; this process is further described in section D.3 of this chapter. The *translate(...)* method returns *true* if the translation is successful and *false* otherwise.

receive(...) – The OOMI Translator will forward the translated document to the destination system, together with the name of the source system.

The application utilizing the OOMI will implement the OOMI Translator according to its specific requirements. Three architectural alternatives are possible, as presented in the next section.

B. OOMI TRANSLATOR IMPLEMENTATION ARCHITECTURAL ALTERNATIVES

As mentioned in the preceding section, the OOMI translator class was meant to be incorporated into a translator (OOMI Translator) that will handle the details of networking between the component systems. In this section, the different architectural alternatives in which the OOMI Translator can be implemented are presented. These alternatives include implementation as a wrapper around either the source system, destination system, or both source and destination systems; or as a standalone server which functions as a middleman for information exchange.

1. OOMI Translator on a Central Server

In the first implementation alternative, each component system communicates with the OOMI Translator running on a central server as depicted in Figure V-3. The OOMI Translator is not required to be implemented on the component systems and each component system only needs to transmit/receive data in their own native format. Of course, a communication protocol must be agreed upon between the central server and the component systems.

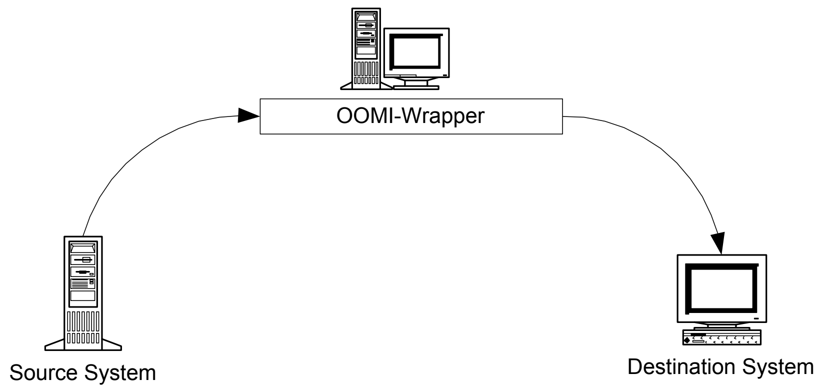


Figure V-3. OOMI Translator located on a central server.

2. OOMI Translator on One Component System

Figure V-4 shows a second architectural alternative where the OOMI Translator is implemented as a wrapper on one of two interoperating systems. Here, two interoperating systems may use their own communication protocol to exchange information between them. The on-the-wire format for information exchanged between systems will be the native format of the system without the OOMI Translator.

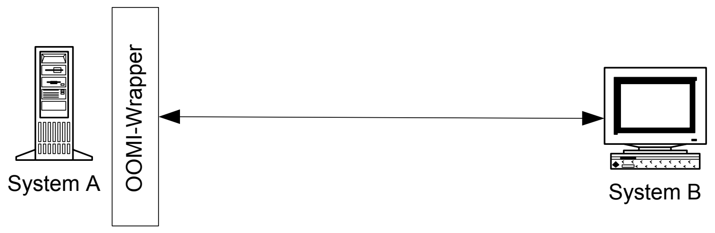


Figure V-4. OOMI Translator on One Component System.

3. OOMI Translator on All Component Systems

The final alternative, depicted in Figure V-5, shows the OOMI Translator implemented as a wrapper around each component system. Information is exchanged between the component systems in an intermediate format internal to the OOMI Translators. The OOMI Translators on each system will need to implement a communication protocol between them.

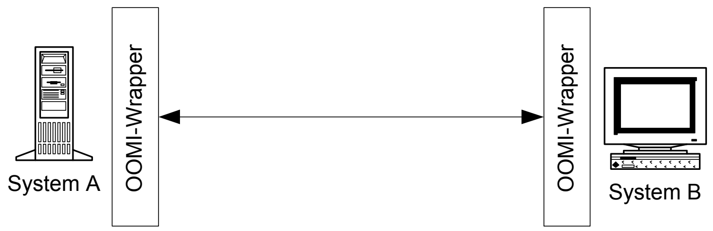


Figure V-5. OOMI Translator on All Component Systems.

C. THE TRANSLATE PROCESS

The translate process identified in Figure V-2 and briefly introduced in section V.A is further detailed in Figure V-6 and discussed below. The sequence diagram depicted in Figure V-6 illustrates the translation process, when the *translate()* method of the OOMI *Translator* object is invoked on a received XML instance document.

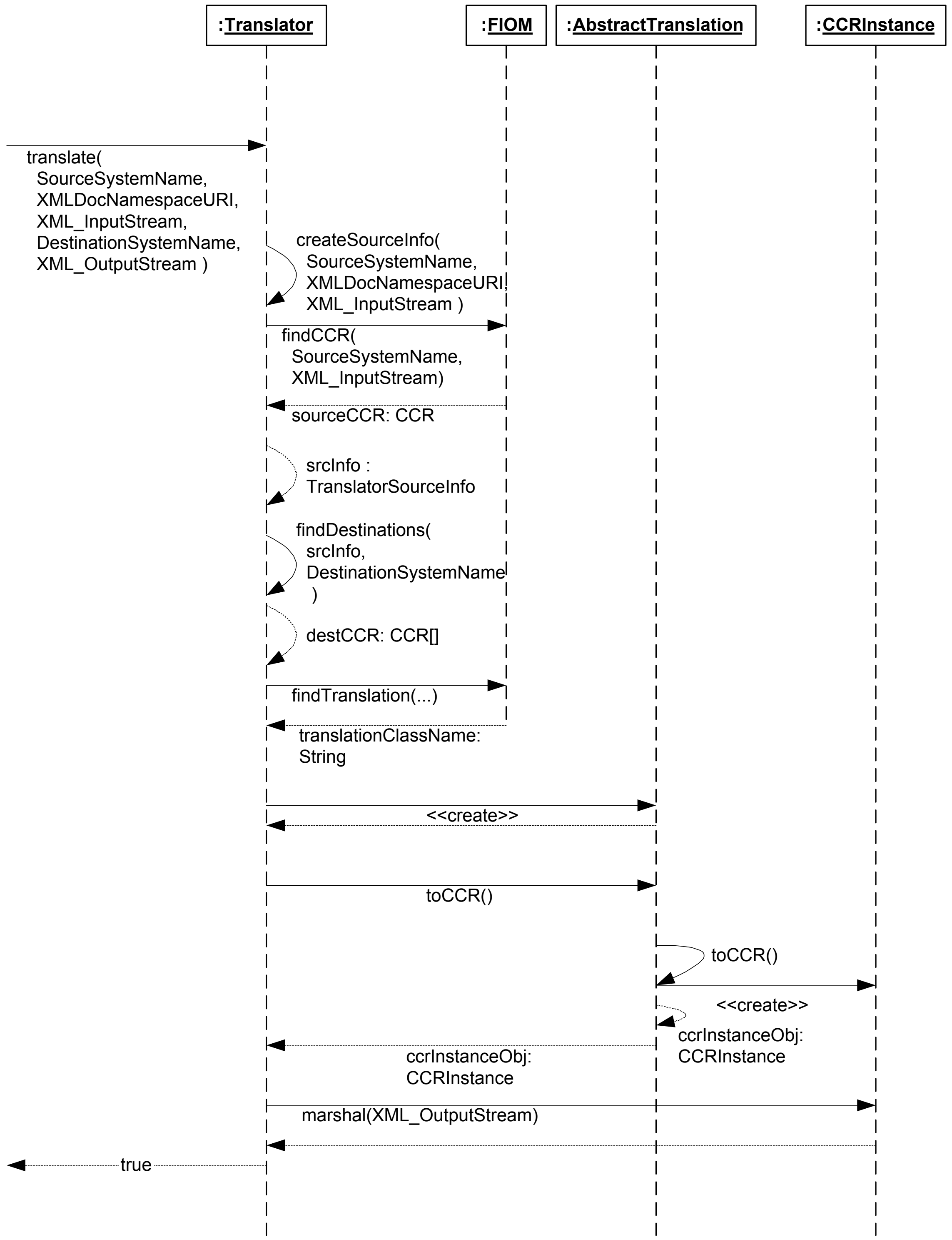

:Translator
:FIOM
:AbstractTranslation
:CCRInstance
translate(SourceSystemName, XMLDocNamespaceURI, XML_InputStream, DestinationSystemName, XML_OutputStream)
createSourceInfo(SourceSystemName, XMLDocNamespaceURI, XML_InputStream)
findCCR(SourceSystemName, XML_InputStream)
sourceCCR: CCR
srcInfo : TranslatorSourceInfo
findDestinations(srcInfo, DestinationSystemName)
destCCR: CCR[]
findTranslation(...)
translationClassName: String
<<create>>
toCCR()
toCCR()
<<create>>
ccrInstanceObj: CCRInstance
ccrInstanceObj: CCRInstance
marshal(XML_OutputStream)
true

Figure V-6. The Translate process.

A discussion of the objects involved and the messages exchanged between them during translation follows.

Translator object – This object is an instance of the *Translator* class, which implements the logic for resolution of differences in views and representations, as specified in requirement *RT-1*.

FIOM object – This object models the FIOM of the federation. It is passed to the *Translator* object when the *Translator* object is created.

CCRInstance object – This object is the result of the translation, it is subsequently marshaled into an XML document.

createSourceInfo(...) – upon receipt of a translate message from the OOMI Translator, the *Translator* object creates a *TranslatorSourceInfo* object containing the source system name, source FCR object, source CCR object and a method to determine if a source FCR attribute has a value. Within this method, the XML document is unmarshaled into its corresponding *CCRInstance* object.

findCCR(...) – the translator invokes the *findCCR(...)* method of the FIOM to list all the CCRs associated with the destination system.

findDestinations() – Returns an array of eligible destination CCRs, in ascending order of eligibility. A destination CCR is considered more eligible than another if it has more of its attributes matched by the source FCR. One of the major capability of the Translator is to automatically determine the "best" destination FCR to use for a destination system. This functionality is implemented in the *findDestinations()* method. The algorithm for determining translation is summarized as follows [Young02]:

> Search the FEV Inheritance Hierarchy containing the received intermediate object's defining FCR Schema Class for an FEV containing a CCR for the destination system. If such a CCR is found, the destination CCR Schema Class is examined to determine if all mandatory properties (attributes and operations) contained in the CCR Schema Class have a corresponding property in the received intermediate object's defining FCR Schema Class.

IF

1. all the mandatory operations have corresponding operations in the received intermediate object's defining FCR Schema Class

AND

2. the mandatory attributes contained in the destination CCR Schema Class have corresponding attributes and the attribute had a value set in the received intermediate object's defining FCR Schema Class

THEN the FCR-CCR Translation class associated with the destination CCR Schema is used to convert the received intermediate FCR Schema Class instance to a destination CCR Schema Class instance.

For the initial prototype implementation, condition one (mandatory operations) is not considered, since this implementation only deals with the attributes.

findTranslation(...) – the translator invokes the *findTranslation(...)* method of the FIOM to determine the class name of the AbstractTranslation class descendent that will perform the translation.

<<create>> – the translator instantiates the specific translation class and invokes the *toCCR()* method, which will perform the translation and returns an instance of *CCRInstance*.

marshal(...) – the translator invokes the *marshal(...)* method of the *CCRInstance* object to convert the *CCRInstance* object into an XML stream.

true – finally, the translator returns *true* indicating a successful translation.

The complete source code listings of the *Translator* class is included in Appendix C.

D. SUMMARY

The OOMI Translator discussed in this chapter is responsible for resolving heterogeneities among systems during runtime to enable interoperation. In this prototype implementation, XML documents are the on-the-wire format exchanged between the component systems.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

A. RESULTS

With the completion of the OOMI Translator and the Translation Generator prototypes in this thesis, it should be clear that the OOMI is achievable and is a practical approach to the *$O(n^2)$* translations problem. In addition to addressing the *$O(n^2)$* translations problem, the OOMI also introduces a systematic way, based on Object-Oriented Analysis and Design (OOAD), in defining and organizing information about real-world entities that can be exchanged between component systems to achieve interoperability.

Efficiency of the translations may be an issue in high-performance systems. Deploying the OOMI Translator directly in Java bytecodes may not be sufficient for such systems. An attractive alternative would be to compile the OOMI Translator and the translation classes it uses into the native machine code on the target platforms of high-performance systems. The fact that the OOMI Translator does not require any graphical user interface makes this approach possible, using compilers such as TowerJ [TowerJ].

B. FUTURE WORK

At the time of this writing, the OOMI toolset is far from being complete. The most crucial factor in the acceptance of any methodology or toolset is the level of usability, automation and integration with other tools. The following areas for future work were identified to be of high priority for the OOMI and its toolset to gain acceptance:

- Two-way editing to support round-trip development
    - The translation generator module currently does not support two-way editing, although provisions had been made within the generated source code to support it.
    - Since software development is an iterative process, seamless two-way editing support is crucial to improve the usability of the OOMI IDE.

- Support for automated testing of translations created through the OOMI IDE.
- Translation code editing/development
    - The OOMI IDE needs to be integrated with a commercial Java IDE to support development of the translation code. The Java IDE should provide the advanced editing and debugging features required by the interoperability engineer for development of any non-trivial translation.
- OOMI Translator and networking proxy and APIs
    - The OOMI toolset should also include a flexible implementation of the OOMI Translator and matching networking APIs that both legacy and newly developed systems can use to achieve interoperability.
    - The OOMI Translator will need to support the common distributed computing frameworks like SOAP, CORBA, DCOM and .Net.
    - An implementation using JINI may be a viable solution, due to JINI's built-in ability to distribute code transparently.
    - The OOMI Translator may also need to address security issues: secrecy, integrity and non-repudiation.
- Supporting real-time systems
    - With the recent release of the reference implementation of Real-Time Specification for Java [RTSJ] Research by TimeSys Corporation [TimeSys], developing true real-time systems in Java is finally possible. Research into adapting the OOMI Translator in real-time Java implementations will benefit the application of OOMI on the integration of real-time systems.

LIST OF REFERENCES

[Young02] Young, P. *Integration of Heterogeneous Software Systems Through Computer-Aided Resolution of Data Representation Differences.* Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, Unpublished.

[CY01] Christie, Brent and Young, Paul. *Integrated Development Environment (IDE) For The Construction Of A Federation Interoperability Object Model (FIOM).* Masters Thesis, Naval Postgraduate School, Monterey, CA. September 2001.

[Young01] Young, P. *Using an Object Oriented Model for Resolving Representational Differences between Heterogeneous Systems.*

[RMI00] Madhusudhan Govindaraju, Aleksander Slominski, Venkatesh Choppella, Randall Bramley, Dennis Gannon. *Requirements for and Evaluation of RMI Protocols for Scientific Computing.* [http://www.extreme.indiana.edu/soap/sc00/paper/index.html] Department of Computer Science, Indiana University, Bloomington, IN.

[XML99] Mark Johnson. *XML JavaBeans ^{TM} Integration, Part 3: Integrate the XMLBeans package with the Java core.* JavaWorld July 1999 [http://developer.java.sun.com/developer/technicalArticles/jbeans/XMLJavaBeans3/]

[WO00] Jeannette M.Wing, John Ockerbloom. *Respectful Type Converters.* IEEE Transactions on Software Engineering, Vol. 26, No. 7, July 2000.

[LW94] Barbara H. Liskov, Jeannette M.Wing. *A Behavioral Notion of Subtyping.* ACM Transactions on Programming Languages and Systems, Vol 16, No 6, Nov 1994, Pages 1811-1841.

[JavaXML] Sun Microsystems Inc. *Understanding XML and the Java XML APIs.* [http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/overview/index.html]

[WebSvc01] Sun Microsystems Inc. *Web Services Made Easier: The Java ^{TM} APIs and Architectures for XML, A Technical White Paper.* [http://java.sun.com/xml/webservices.pdf]

[Melnik+99] Sergey Melnik et al. *Introducing the Generic Interoperability Framework, Working Draft, 1999.* [http://www-diglib.stanford.edu/diglib/ginf/WD/ginf-overview/]

[Melnik+] Sergey Melnik et al. *Generic Interoperability Framework (GINF) Middleware.* [http://www-diglib.stanford.edu/diglib/ginf/WD/ginf-

	magic/]
[Wie93]	Wiederhold, G., *Intelligent Integration of Information*, ACM-SIGMOD 93, Washington, DC, May 1993, pp. 434-437.
[HLA99]	Judith S. Dahmann, James O. Calvin, and Richard M. Weatherly. *A Reusable Architecture for Simulations.* Communications of the ACM, Sep 1999 / Vol. 42, No. 9, Pages 79-84.
[Sun01]	Reinhold, M., *The Java ™ Architecture for XML Binding (JAXB), Working-Draft Specification*, Sun Microsystems, Inc., [ftp://ftp.java.sun.com/pub/xml/987dfjaxb10ea3ds/jaxb-0_21-wd-spec.pdf]. 30 May 2001.
[Hadfield]	Jeff Hadfield. *Microsoft's BizTalk Initiative: Do They Really Have the XML Religion?* [http://www.devx.com/free/newsletters/xml/ednote0607.asp]
[Travis]	Brian E. Travis. *XML and SOAP Programming for BizTalk Servers.* Microsoft Press.
[BizTalk]	*BizTalk Home Page*. [http://www.biztalk.org]
[TowerJ]	*TowerJ Home Page*. [http://www.towerj.com/]
[RTSJ]	*RTJ.org*. [http://www.rtj.org/]
[TimeSys]	*TimeSys Website*. [http://www.timesys.com/rtj/index.html]

APPENDIX A. FIOM FRAMEWORK SOURCE CODE

A. PACKAGE: mil.navy.nps.cs.oomi

1. OOMIDatabase.java

```
package mil.navy.nps.cs.oomi;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

/**
 * This interface represents the database that contains
 * the FIOMs maintained by the OOMI IDE,
 * each of these FIOMs can be used by the OOMI Translator.
 * It serves to encapsulate the access of persistent
 * storage mechanisms.
 *
 * @author LSC
 * @version 1.0
 */

public interface OOMIDatabase {

  /**
   * Returns an array of FIOMs already registered in this OOMIDatabase.
   * @return Returns an array of FIOMs already registered in this OOMIDatabase.
   *         Returns an empty array if no FIOM's are registered.
   */
  public FIOM[] getFIOM() ;

  /**
   * Returns the number of FIOM's already registered in this database.
   */
  public int fiomCount();

  /**
   * Creates a new FIOM with the specified name
   * and adds it to this OOMIDatabase.
   * Returns a the newly created instance of FIOM .
   *
   * @param fiomName A unique FIOM name for the new FIOM.
   * @return Returns a new instance of a FIOM with the specified name.
   * @throws DuplicateKey Throws DuplicateKey if a FIOM with the same name
   *                      already exists in this OOMIDatabase.
   */
  public FIOM newFIOM( String fiomName ) throws DuplicateKeyException;

  /**
   * Finds a FIOM identified by the specified name.
   * @param fiomName The name of the FIOM to be returned.
   * @return Returns FIOM instance identified by the specified name,
   *         returns null if no FIOM
   *         with matching (exact, case-insensitive) name is found.
   */
  public FIOM findFIOM( String fiomName );

  /**
   * Returns if there is a FIOM with the  specified name in this database.
   */
  public boolean fiomExists( String fiomName );
}
```

2. SearchHandler.java

```
package mil.navy.nps.cs.oomi;

/**
 * @author LEE ShongCheng
 * @version 1.0
 */

public interface SearchHandler {

  /**
   * Indicates whether to find more.
   */
  public boolean carryOn();

  /**
   * Indicates whether the specified object satisfies this SearchCriteria.
   */
  public boolean satisfied(Object obj);

  /**
   * Adds an object that satisfies this criteria.
   */
  public void add(Object obj);

}
```

B. PACKAGE: mil.navy.nps.cs.oomi.fiom

1. AbstractTranslation.java

```
package mil.navy.nps.cs.oomi.fiom;

import mil.navy.nps.cs.oomi.fiom.FCRInstance;
import mil.navy.nps.cs.oomi.fiom.CCRInstance;
//import mil.navy.nps.cs.oomi.impl.*;

/**
 * @author LSC
 * @version 1.0
 */

public abstract class AbstractTranslation {

  /**
   * The FCR that this translator supports.
   */
  static private Class _fcrInstanceClass;
  /**
   * The CCR that this translator supports.
   */
  static private Class _ccrInstanceClass;

  /**
   * Initializes this class with the corresponding FCR and CCR.
   * It is recommended that fcr and ccr, use fully qualified class names.
   */
  protected static void init( Class fcrInstanceClass, Class ccrInstanceClass ) {
    setFCRInstanceClass(fcrInstanceClass);
    setCCRInstanceClass(ccrInstanceClass);
  }
  /**
   * Translates to FCR.
   */
  abstract public FCRInstance toFCR( CCRInstance ccr );

  /**
   * Translates to CCR.
   */
  abstract public CCRInstance toCCR( FCRInstance fcr );
  /**
   * Sets the FCR that this translator supports.
   */
  protected static void setFCRInstanceClass( Class fcrInstanceClass ) {
    _fcrInstanceClass = fcrInstanceClass;
  }
  /**
   * Sets the CCR that this translator supports.
   */
  protected static void setCCRInstanceClass( Class ccrInstanceClass ) {
    _ccrInstanceClass = ccrInstanceClass;
  }
  /**
   * Returns the FCR that this translator supports.
   */
  public static Class getFCRInstanceClass() {
    return _fcrInstanceClass;
  };

  /**
   * Returns the CCR that this translator supports.
   */
  public static Class getCCRInstanceClass() {
    return _ccrInstanceClass;
  };


}
```

2. Attribute.java

```
package mil.navy.nps.cs.oomi.fiom;
```

```
import java.util.List;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

/**
 * Contains the information of a FCR or CCR class attribute.
 * If the Attribute is an array, that is, isArray()==true,
 * the getType() returns the type of the
 * element in the array.
 * @author LSC
 * @version 1.0
 */

public interface Attribute  {

  /**
   * The standard separator for the name path to an Attribute
   * that is a descendent of another Attribute.
   */
  public final String NAME_SEPARATOR = ".";

  /**
   * Creates and adds a new child attribute for this object.
   */
  public Attribute newChild( String attributeName, TypeName attributeType)
                       throws UniqueNameViolationException ;

  /**
   * Removes the specified descendent from this attribute,
   * there should only be one instance of any attribute within the hierarchy,
   * the parent field of the descendent is cleared.
   * @param descendentToRemove The descendent to be removed.
   * @return Returns true if a descendent had been removed,
   *         false if no descendent is removed, that is, the descendent is
   *         not found within this Attribute..
   */
  public boolean removeDescendent( Attribute descendentToRemove );

  /**
   * Creates an exact copy of this Attribute.
   */
```

```
public Attribute copy();

/**
 * Returns the number of children of this object.
 */
public int childCount();

/**
 * Returns the name of this attribute.
 */
public String getName();
/**
 * Sets the name of this attribute.
 */
public void setName(String name);

/**
 * Sets the type of this Attribute.
 */
public void setType (TypeName type);

/**
 * Returns the type of this Attribute.
 */
public TypeName getType();

/**
 * Returns the parent of this attribute, returns null if this attribute has
 * no parent, that is, it is the root attribute.
 * @return Returns the parent of this attribute, returns null if this attribute has
 *         no parent, that is, it is the root attribute.
 */
public Attribute getParent();

/**
 * Returns the children (grandchildren excluded) of this attribute.
 */
public Attribute[] getChild();

/**
 * Finds a child attribute matching (not case-sensitive) the specified name.
 */
public Attribute findChild(String childName);
```

```
/**
 * Finds a descendent attribute who is equal to the specified Attribute attr.
 */
public Attribute findDescendent(Attribute attr);

/**
 * Enumerates all descendents and add them to the specified list,
 * effectively flattening the attribute tree into a list.
 */
public void enumerateDescendents( List result );

/**
 * Returns an array of all ancestors of this Attribute, from
 * greatest grandparent to this Attribute.
 * @return Returns an array of all ancestors of this Attribute, from
 *         greatest grandparent to this Attribute.
 */
public Attribute[] pathToArray();

/**
 * Determines if the given Attribute is equal to this object.
 * @return Returns true if and only if (
 *                          this.getName().equals(attr.getName())
 *                       && this.getType().equals(attr.getType())
 *                       && this.getParent().equals(attr.getParent())
 *                       )
 */
public boolean equals( Object attrObj );

/**
 * As required by the general contract of Object.hashCode(),
 * implementations of Attribute shall ensure that for any Attribute a1, a2,
 * a1.equals(a2) implies a1.hashCode()==a2.hashCode().
 */
public int hashCode();

/**
 * Returns true if this attribute is mandatory, false otherwise.
 * @return Returns true if and only if getMinOccurs()>=1
 */
public boolean isMandatory();
```

```
  /**
   * Returns the full name path of this Attribute within the Attribute tree.
   */
  public String fullPath();

  /**
   * Returns true if this attribute is an array type.
   *
   * @return Returns true if this attribute is an array type.
   */
  public boolean isArray();

  public int getMinOccurs();
  public void setMinOccurs( int v );
  public int getMaxOccurs();
  public void setMaxOccurs( int v );
}
```

3. AttributeMapping.java

```
package mil.navy.nps.cs.oomi.fiom;

/**
 * @author LSC
 * @version 1.0
 */

public interface AttributeMapping {

  public Attribute[] fromAttributes();

  public Attribute toAttribute();
}
```

4. CastorHelper.java

```
package mil.navy.nps.cs.oomi.fiom;

import java.lang.reflect.*;
import java.beans.*;
import java.util.List;
import java.util.ArrayList;

import org.exolab.castor.mapping.*;
import org.exolab.castor.xml.*;

/**
 * Helper class for Castor related operations.
 */
public class CastorHelper {

  /**
   * Not instantiable.
   */
  private CastorHelper() {
  }

  /**
   * Builds an array of Attribute objects based on the specified Class.
   * <br>
   * The specified Class should be generated by Castor from an XML schema.
   * Only subclasses of FCRInstance or CCRInstance will be expanded with
   * sub-attributes,
   * but indexed property whose element is a subclass of
   * FCRInstance or CCRInstance will NOT be expanded.
   * <br>
   * To add the elements in the array returned to a Schema object, just call
   * Schema.addAttribute() for each element.
   */
  public static Attribute[] classAttributes(
                                    FIOMFactory factory,
                                    XMLClassDescriptor classDescriptor
                                    ) throws ClassNotFoundException,
                                             IllegalAccessException,
                                             InstantiationException {
    return classAttributes( factory, classDescriptor, null );
  }



  /**
   * Returns an array of Attribute instances, with the specified Attribute, parent,
   * as the parent.
   */
   public static Attribute[] classAttributes( FIOMFactory factory,
                                              XMLClassDescriptor classDescriptor,
                                              Attribute parent )
                                              throws ClassNotFoundException,
                                                    IllegalAccessException,
                                                    InstantiationException {
    XMLFieldDescriptor[] fieldDescs = classDescriptor.getElementDescriptors();
    if (fieldDescs==null)
      return new Attribute[0];
    Attribute[] result = new Attribute[fieldDescs.length];
    XMLFieldDescriptor field;
    FieldValidator validator;
    Class fieldType;
    TypeName typeName;
    for (int i=0; i<result.length; i++) {
      field = fieldDescs[i];
      validator = field.getValidator();
      fieldType = field.getFieldType();
      if (field.isMultivalued()) {
        typeName = factory.makeArrayTypeName(
                            fieldType,
                            1 );
      }
      else {
        typeName = factory.makeTypeName(fieldType);
      }
      result[i] = factory.makeAttribute( parent, decapitalize(field.getXMLName()), typeName );

      result[i].setMinOccurs( validator.getMinOccurs() );
      result[i].setMaxOccurs( validator.getMaxOccurs() );
      //p(i + " AttrName: " + result[i].getName());
      if (   FCRInstance.class.isAssignableFrom(fieldType)
          || CCRInstance.class.isAssignableFrom(fieldType) ) {
        Class c = Class.forName(fieldType.getName()+"Descriptor");
        XMLClassDescriptor classDesc = (XMLClassDescriptor) c.newInstance();
        classAttributes( factory,
                         classDesc,
                         result[i] );
      }
    }
    return result;
  }

  /**
   * Decapitalize the first character, unless the 2nd character is also in uppercase.
   */
  public static String decapitalize( String s ) {
    if (s.length()==0)
      return s;
    else {
      if (s.length()>1)
        if (s.charAt(1)==Character.toUpperCase(s.charAt(1)))
          return s;
        else
          return Character.toLowerCase(s.charAt(0)) + s.substring(1);
      else
        return s.toLowerCase();
    }
  }
  /**
   * Build up the Attribute tree for the specified PropertyDescriptor.
   * Only subclasses of FCRInstance or CCRInstance will be expanded with
   * sub-attributes.
   * An indexed property whose element is a subclass of
   * FCRInstance or CCRInstance will NOT be expanded.
   */
  public static Attribute propertyToAttribute(
                                    FIOMFactory factory,
                                    Attribute parent,
                                    PropertyDescriptor desc )
                                    throws IntrospectionException {
    TypeName typeName;
    Class descType;
    if (desc instanceof IndexedPropertyDescriptor) {
      descType = ((IndexedPropertyDescriptor)desc).getIndexedPropertyType();
      typeName = factory.makeArrayTypeName(
                          descType,
                          1 );
    }
    else {
      descType = desc.getPropertyType();
      typeName = factory.makeTypeName(descType);
    }
    Attribute result = factory.makeAttribute( parent, desc.getName(), typeName );
    if (   FCRInstance.class.isAssignableFrom(descType)
        || CCRInstance.class.isAssignableFrom(descType) ) {
      PropertyDescriptor[] childs = castorClassProperties( descType );
      for (int i=0; i<childs.length; i++) {
        propertyToAttribute( factory, result, childs[i] );
      }
    }
    return result;
  }

  /**
   * Returns an array of PropertyDescriptor for
   * the specified class.
   * The properties "valid" (Castor specific property)
   * and "class" (property common to all classes) are not returned.
   */
  public static PropertyDescriptor[] castorClassProperties(  Class aClass )
                                          throws IntrospectionException {
    return classProperties( aClass, new String[] {"class", "valid"} );
  }
  /**
   * Returns an array of PropertyDescriptor for
   * the specified class.
   * Properties with names found in namesToIgnore are not included in the
   * array returned.
   */
  public static PropertyDescriptor[] classProperties(  Class aClass, String[] namesToIgnore )
                                          throws IntrospectionException {
    BeanInfo beanInfo = java.beans.Introspector.getBeanInfo(aClass);
    PropertyDescriptor[] propDescs = beanInfo.getPropertyDescriptors();
    List result = new ArrayList(propDescs.length-1);
    String attrName;
    boolean ignore;
    for (int i=0; i<propDescs.length; i++) {
      attrName = propDescs[i].getName();
      ignore = false;
      for (int j=0; j<namesToIgnore.length; j++) {
        if (attrName.equals(namesToIgnore[j]))
         ignore = true;
      } // for
      if (!ignore)
        result.add(propDescs[i]);
    }
    return (PropertyDescriptor[]) result.toArray( new PropertyDescriptor[result.size()] );
  }

  public static void p(String msg) {
    System.out.println(msg);
  }
```

```
}
```

5. CCR.java

```
package mil.navy.nps.cs.oomi.fiom;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;



/**
 * Represents a CCR in OOMI.
 * A CCR should be uniquely defined by:
 *  <ol>
 *    <li> SystemName and CCRName
 *    <li> SystemName and XMLNameSpaceURI
 *  </ol>
 * within the FIOM.
 * <p>
 * Responsibilities:
 * <ul>
 * <li> Represents a CCR in the OOMI FIOM.
 * <li> Maintain information on whether an attribute of its FCR is mandatory,
 *      based on the FCRtoCCR attribute mappings contained in this object.
 * </ul>
 *
 * @author LSC
 * @version 1.0
 */

public interface CCR {

  /**
   * Returns the name of this CCR.
   */
  public String getCCRName();

  /**
   * Returns the name of the system this CCR is defined for.
   */
  public String getSystemName();
```

```
/**
 * Returns the FCR of this CCR.
 */
public FCR getFCR() ;

/**
 * Joins this CCR to the specified FCR.
 * If bewFCR==null, nothing is done.
 * @param newFCR The FCR that this CCR will be joining.
 */
public void joinFCR( FCR newFCR ) ;

/**
 * Returns the CCRSchema for this object.
 */
public CCRSchema getCCRSchema();

/**
 * Returns the fully-qualified Castor
 * generated Java class's (a descendent of CCRInstance) name for this CCR.
 * A String is used instead of Class so that it may be possible
 * to work with the CCR without actually loading the bytecode for the
 * class.
 */
public String getJavaClassName();

/**
 * Sets the fully-qualified name of the Castor
 * generated Java class (a descendent of CCRInstance) for this CCR.
 */
public void setJavaClassName( String javaClassName );

/**
 * Returns the XML Schema URI this CCR is defined for.
 */
public String getXMLNameSpaceURI();

/**
 * Sets the XML Schema URI this CCR is defined for.
 * A well-formed valid URI is expected, no error checking is performed.
 */
public void setXMLNameSpaceURI( String xmlNameSpaceURI );
```

```
  /**
   * The given FCR attribute is mandatory if it is mapped to one or more mandatory
   * CCR attributes.
   *
   */
  public boolean isMandatoryFCRAttribute( Attribute fcrAttr );

  /**
   * Returns an array of mandatory attributes of the FCR this CCR is associated
   * with.
   */
  public Attribute[] mandatoryFCRAttributes();
  /**
   * Returns the number of mandatory attributes of the FCR this CCR is associated
   * with.
   */
  public int mandatoryFCRAttributeCount();
  /**
   * Adds a mapping of FCR attributes to the attribute of this CCR object.
   */
  public void addFCRToCCRAttributeMapping( Attribute[] from, Attribute to );

  /**
   * Clears the FCR to CCR attribute mappings.
   */
  public void clearFCRToCCRAttributeMappings();


}
```

6. CCRInstance.java

```
package mil.navy.nps.cs.oomi.fiom;



import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;

import org.exolab.castor.xml.*;
import org.exolab.castor.xml.MarshalException;
import org.exolab.castor.xml.ValidationException;

import mil.navy.nps.cs.oomi.exceptions.*;

/**
 * Base class for all marshalled CCR Instances, it abstracts the marshalling
 * and unmarshalling to/from XML, all Castor generated classes from the
 * component systems' schemas should use this class as their base class.
 * <br/>
 *
 * @author LSC
 * @version 1.0
 */

abstract public class CCRInstance {
  /**
   *
   */
  public static CCRInstance unmarshal(Class theClass, java.io.InputStream inStream)
                                  throws UnmarshalException  {
    try {
      return (CCRInstance) Unmarshaller.unmarshal(
                                            theClass,
                                            new InputStreamReader(inStream) );
    }
    catch (Exception exc) {
      throw new UnmarshalException( exc.getClass().getName() + ": " + exc.getMessage() );
    }
  };
  public void marshal(java.io.OutputStream outStream)
                                  throws ValidationException,
                                         MarshalException  {
    Marshaller.marshal(this, new OutputStreamWriter(outStream));
  }
}
```

7. CCRSchema.java

```
package mil.navy.nps.cs.oomi.fiom;

/**
 * @author LSC
 * @version 1.0
 */

public interface CCRSchema extends Schema {

}
```

8. CCRSemantics.java

```
package mil.navy.nps.cs.oomi.fiom;

/**
 * @author LSC
 * @version 1.0
 */

public interface CCRSemantics {

}
```

9. CCRSyntax.java

```
package mil.navy.nps.cs.oomi.fiom;

/**
 * @author LSC
 * @version 1.0
 */
public interface CCRSyntax {

}
```

10. FCR.java

```
package mil.navy.nps.cs.oomi.fiom;

import java.net.URL;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;


/**
 * @author LSC
 * @version 1.0
 */

public interface FCR {

  /**
   * Returns the name of this FCR.
   */
  public String getFCRName();

  /**
   * Returns the FEV of this FCR.
   */
  public FEV getFEV();

  /**
   * Returns the number of CCRs defined for this FCR.
   */
  public int ccrCount();

  /**
   * Returns all the CCRs defined for this FCR in an array.
   * @return Returns all the CCRs defined for this FCR in an array,
   *         an empty array is returned if no CCRs defined.
   */
  public CCR[] getCCR();



  /**
   * Finds all the CCRs which satisifies the specified criteria.
   * @param criteria
   */
  public void findCCR(SearchHandler handler);

  /**
   * Finds all the CCR with for the specified system.
   * @param systemName
   * @return Returns all the CCRs in this FCR that belongs to the specified system name
   *         (not case-sensitive).
   *         Returns an empty array if not found.
   */
  public CCR[] findCCR( String systemName );
```

```
/**
 * Finds the CCR with the specified name.
 * @param systemName
 * @param ccrName
 * @return Returns the CCR in this FCR that has the specified name
 *         (not case-sensitive).
 *         Returns null if not found.
 */
public CCR findCCR( String systemName, String ccrName );

/**
 * Finds the CCR with the specified system name and xmlNameSpaceURI.
 * @param systemName
 * @param xmlNameSpaceURI
 * @return Returns the CCR in this FCR that has the specified system name
 *         (not case-sensitive) and xmlNameSpaceURI.
 *         Returns null if not found.
 *         xmlNameSpaceURI==null shall always result in a return value of null.
 */
public CCR findCCR( String systemName, URL xmlNameSpaceURI );

/**
 * Returns true if this FCR has a CCR with specified name.
 */
public boolean ccrExists(String systemName, String ccrName);

/**
 * Returns true if this FCR has a CCR with specified name.
 * @return Returns true if this FCR has a CCR with specified name.
 *         Returns false if xmlNameSpaceURI==null.
 */
public boolean ccrExists(String systemName, URL xmlNameSpaceURI);


/**
 * Creates a new instance of CCR for this FCR and returns the newly created
 * instance.
 * @param systemName The system name.
 * @param ccrName The name of the new CCR.
 * @param xmlNameSpaceURI The URL to the XML schema that the new CCR represents.
 *                    Pass in null if XML schema is not required.
 *
```

```
   * @throws DuplicateKeyException When the specified systemName,ccrName pair or
   *                               systemName, xmlNameSpaceURI pair
   *                               already exist for this FCR.
   */
  public CCR newCCR(String systemName, String ccrName, URL xmlNameSpaceURI)
                                    throws DuplicateKeyException;



  /**
   * Returns the FCRSchema for this object.
   */
  public FCRSchema getFCRSchema();

  /**
   * Returns the Java class name for this FCR.
   */
  public String getJavaClassName();

  /**
   * Sets the Java class name for this FCR.
   */
  public void setJavaClassName(String className);

  /**
   * Finds the CCR that represents the specified XML schema.
   * A case-sensitive comparison is performed.
   * @param xmlNameSpaceURI The URL to the XML schema whose corresponding CCR
   *                     is to be found.
   * @return Returns the CCR that represents the specified XML schema
   *         corresponding to this FCR.
   *         Returns null if no such CCR exists or if xmlNameSpaceURI==null.
   */
//  public CCR findCCR( URL xmlNameSpaceURI );

}
```

11. FCRInstance.java

```
package mil.navy.nps.cs.oomi.fiom;
/**
 * Base class for all FCR Instances.
 * <br/>
 *
 * @author LSC
 * @version 1.0
 */
public class FCRInstance {
  public FCRInstance() {
  }
}
```

12. FCRInstanceHelper.java

```
package mil.navy.nps.cs.oomi.fiom;

import java.util.*;
import java.lang.reflect.*;
import java.beans.*;
import java.lang.ref.SoftReference;



/**
 * @author LSC
 * @version 1.0
 */

public class FCRInstanceHelper {

  protected FCRInstanceHelper() {
  }
  /**
   * Clones srcObj,
   * returning a new FCRInstance object of type dstClass,
   * only shallow copy is performed for all properties found in both dstClass
   * and srcObj.
   */
  public static FCRInstance clone( FCRInstance srcObj, Class dstClass )
                                    throws IntrospectionException,
                                           IllegalAccessException,
                                           InstantiationException,
                                           InvocationTargetException,
                                           NoSuchMethodException {
    PropertyDescriptor[] srcPropDescs = classProperties(srcObj.getClass());
    HashMap srcPropMap = toHashMap(srcPropDescs);
    PropertyDescriptor[] dstPropDescs = classProperties(dstClass);
    HashMap dstPropMap = toHashMap(dstPropDescs);
    FCRInstance result = (FCRInstance)dstClass.newInstance();
    PropertyDescriptor dstDesc;
    PropertyDescriptor srcDesc;
    String propName;
    for (int i=0; i<srcPropDescs.length; i++) {
      srcDesc = srcPropDescs[i];
      propName = srcDesc.getName();
      dstDesc = (PropertyDescriptor) dstPropMap.get(propName);
      if (dstDesc!=null)
        if ( hasValue( srcObj, propName ) )
          // write the value only if the has<PropertyName> value in srcObj is true
          writeValue( dstDesc, result, readValue(srcDesc, srcObj) );
    }
    return result;
  }

  /**
   * Clones srcObj,
   * returning a new FCRInstance object of type dstClass,
   * only shallow copy is performed for all properties found in both dstClass
   * and srcObj.
   */
  public static FCRInstance clone1( FCRInstance srcObj, Class dstClass )
                                    throws IntrospectionException,
                                           IllegalAccessException,
                                           InstantiationException,
                                           InvocationTargetException {
    PropertyDescriptor[] srcPropDescs = classProperties(srcObj.getClass());
    HashMap srcPropMap = toHashMap(srcPropDescs);
    PropertyDescriptor[] dstPropDescs = classProperties(dstClass);
    HashMap dstPropMap = toHashMap(dstPropDescs);
    FCRInstance result = (FCRInstance)dstClass.newInstance();
    PropertyDescriptor dstDesc;
    PropertyDescriptor srcDesc;
    String propName;
    for (int i=0; i<srcPropDescs.length; i++) {
      srcDesc = srcPropDescs[i];
      propName = srcDesc.getName();
      dstDesc = (PropertyDescriptor) dstPropMap.get(propName);
      if (dstDesc!=null)
        if ( hasValue( srcObj, srcPropMap, srcDesc ) )
          // write the value only if the has<PropertyName> value in srcObj is true
          writeValue( dstDesc, result, readValue(srcDesc, srcObj) );
    }
    return result;
  }

  /**
   * Writes the property value for the specified object into the property described
   * by the specified PropertyDescriptor.
   * <br>
   * If the property is readonly, that is, no setter method for the specified
   * PropertyDescriptor exists, nothing is done.
   * @param desc The PropertyDescriptor for the object whose property is be changed.
   * @param obj The object whose property is to be changed.
   * @param value The new value of the property.
   */
  static void writeValue( PropertyDescriptor desc, Object obj, Object value )
                                      throws InvocationTargetException,
                                             IllegalAccessException  {
    Method writeMethod = desc.getWriteMethod();
    // if property is readonly, do nothing.
    if (writeMethod!=null) {
      Object[] args = { value };
      writeMethod.invoke( obj, args );
    }
  }
  /**
   * Reads the property value for the specified object into the property described
   * by the specified PropertyDescriptor.
   * @param desc The PropertyDescriptor for the object whose property is be read.
   * @param obj The object whose property is to be read.
   * @return Returns the value of the specified object's specified property's value.
   */
  static Object readValue( PropertyDescriptor desc, Object obj )
                                      throws InvocationTargetException,
                                             IllegalAccessException  {
    Method readMethod = desc.getReadMethod();
    Object[] args = {};
    return readMethod.invoke( obj, args );
  }
  /**
   * Returns an array of PropertyDescriptor for
   * the specified class of FCRInstance, the getClass() method present in all
   * objects is ignored.
   */
```

```
static PropertyDescriptor[] classProperties( Class aClass )
                                        throws IntrospectionException {
  BeanInfo beanInfo = Introspector.getBeanInfo(aClass);
  PropertyDescriptor[] propDescs = beanInfo.getPropertyDescriptors();
  List result = new ArrayList(propDescs.length-1);
  String attrName;
  for (int i=0; i<propDescs.length; i++) {
    attrName = propDescs[i].getName();
    if (attrName.equals("class"))
      continue;
    //p( attrName  +  "  " + propDescs[i].getPropertyType().getName());
    result.add(propDescs[i]);
  }
  return (PropertyDescriptor[]) result.toArray( new PropertyDescriptor[result.size()] );
}
/**
 * Returns an array of property names for the specified FCRInstance.
 */
public static String[] extractAttributeNames( FCRInstance obj )
                            throws IntrospectionException {
  PropertyDescriptor[] descs = classProperties(obj.getClass());
  String[] result = new String[descs.length];
  for (int i=0; i<descs.length; i++) {
    result[i] = descs[i].getName();
  }
  return result;
}
/**
 * Creates a PropertyDescriptor HashMap using the Property name as the key.
 */
static HashMap toHashMap( PropertyDescriptor[] descs ) {
  HashMap map = new HashMap();
  for (int i=0; i<descs.length; i++) {
    map.put( descs[i].getName(), descs[i] );
  }
  return map;
}

/**
 * Capitalizes the first character of the specified string.
 */
static String capitalize(String s) {
  if (s==null) return null;
  if (s.length()>0)
    return Character.toUpperCase(s.charAt(0)) + s.substring( 1, s.length() );
  else
    return s;
}



/**
 *  A HashMap to cache PropertyDescriptor arrays.
 */
static Map _propDescArrayMap = new WeakHashMap();

static Map _propDescMap = new WeakHashMap();

/**
 * Returns an instance of PropertyDescriptor[] for the specified Object,
 * uses _propDescArrayMap if possible.
 * If the required array is not found in the map, the array
 * is created and added to the map.
 * @param obj
 * @param attr
 * @return An array of PropertyDescriptor for the specified Object.
 */
static PropertyDescriptor[] propertyDescriptorArray(Object obj)
                                throws IntrospectionException {
  String key = obj.getClass().getName();
  PropertyDescriptor[] result = (PropertyDescriptor[])_propDescArrayMap.get(key);
  if (result==null) {
    result = classProperties( obj.getClass() );
    _propDescArrayMap.put( key, result );
  }
  return result;
}
/**
 * Returns an instance of PropertyDescriptor for the specified Object,
 * uses _propDescMap if possible.
 * If the required array is not found in the map, the array
 * is created and added to the map.
 * @param obj
 * @param propertyName
 * @return A PropertyDescriptor of the property with the specified name
 *         of the specified Object.
 */
```

```
static PropertyDescriptor propertyDescriptor( Object obj,
                                              String propertyName)
                                              throws IntrospectionException {
  String key = obj.getClass().getName() + "." + propertyName;
  PropertyDescriptor result = (PropertyDescriptor)_propDescMap.get(key);
  if (result==null) {
    PropertyDescriptor[] descArray = propertyDescriptorArray(obj);
    PropertyDescriptor desc;
    for (int i=0; i<descArray.length; i++) {
      desc = descArray[i];
      if (propertyName.equals(desc.getName()))
        result = desc;
      // cache all descriptors, since it is expected that all descriptors for
      // the object will be required in the near future
      _propDescMap.put( key, result );
    }
  }
  return result;
}

/**
 * Finds the descriptor in the specified PropertyDescriptor array with the
 * name as specified in the parameter name.
 *
 */
//static PropertyDescriptor findDescriptor( PropertyDescriptor

/**
 *
 */
public static Method hasserMethod( Class cls, String fieldName )
                        throws NoSuchMethodException{
  String methodName = "has" + capitalize(fieldName);
  return cls.getMethod( methodName, null );
}
/**
 *  If the specified Object does not have a "has" method for the specified
 *  fieldname, a true value is returned.
 */
public static boolean hasValue( Object obj, String fieldName )
                                throws NoSuchMethodException,
                                       InvocationTargetException,
                                       IllegalAccessException {
```

```
  try {
    Object result = hasserMethod( obj.getClass(), fieldName ).invoke( obj, null );
    return ((Boolean)result).booleanValue();
  }
  catch (NoSuchMethodException e) {
    return true;
  }
}
/**
 * Indicates if the field refered to by the specified Attribute
 * of the specified FCRInstance has a value set.
 * @param object
 * @param attr
 * @return true if the specified Attribute's value had been set,
 *         false if the Attribute value had not been set.
 */
public static boolean hasValue( FCRInstance object, Attribute attr )
                         throws InvocationTargetException,
                                IllegalAccessException,
                                NoSuchMethodException,
                                IntrospectionException {
  Attribute[] path = attr.pathToArray();
  FCRInstance currObj = object;
  Attribute curr;
  String currPropertyName;
  boolean hasValue = false;
  for (int i=0; i<path.length; i++) {
    curr = path[i];
    currPropertyName = curr.getName();
    hasValue = hasValue( currObj, currPropertyName );
    if (!hasValue)
      // if any ancestor does not have a value, then the attribute
      // should not have a value as well
      return false;
    if (i<path.length-1)
      // last attribute
      currObj = (FCRInstance) readValue( propertyDescriptor(currObj, currPropertyName),
                                       currObj );
  }
  return hasValue;
}

/**
```

```
 * Indicates if the property refered to by the specified Attribute
 * of the specified FCRInstance has a value set.
 * @param object
 * @param attr
 * @return A PropertyDescriptor instance for the specified Attribute of the
 *         specified FCRInstance.
 */
public static boolean hasPropertyValue( FCRInstance object, Attribute attr )
                         throws InvocationTargetException,
                                IllegalAccessException,
                                IntrospectionException {
  Attribute[] path = attr.pathToArray();
  FCRInstance currObj = object;
  Attribute curr;
  String currPropertyName;
  boolean hasValue = false;
  for (int i=0; i<path.length; i++) {
    curr = path[i];
    currPropertyName = curr.getName();
    hasValue = hasPropertyValue( currObj, currPropertyName );
    if (!hasValue)
      // if any ancestor does not have a value, then the attribute
      // should not have a value as well
      return false;
    if (i<path.length-1)
      // last attribute
      currObj = (FCRInstance) readValue( propertyDescriptor(currObj, currPropertyName),
                                       currObj );
  }
  return hasValue;
}


/**
 * Indicates whether a property with the specified propertyName
 * has a value set, by inspecting
 * the boolean value of the has<PropertyName> property in propDescs.
 * If no such method is found for the property, a true value is returned.
 * @param obj
 * @param propDesc
 * @param propertyName
 * @return True if the property of the specified Object parameter's specified
 *         PropertyDescriptor parameter has a value set.
 *
 */
static boolean hasPropertyValue( Object obj, String propertyName )
                            throws IllegalAccessException,
                                   InvocationTargetException,
                                   IntrospectionException {
  String hasPropertyName = "has" + capitalize( propertyName );
  PropertyDescriptor hasPropertyDesc
        = (PropertyDescriptor) propertyDescriptor(obj, hasPropertyName);
  if (hasPropertyDesc!=null) {
    Class hasPropertyType = hasPropertyDesc.getPropertyType();
    if (hasPropertyType==Boolean.TYPE) {
      return ((Boolean) readValue( hasPropertyDesc, obj )).booleanValue();
    }
  }
  return true;
}


/**
 * Indicates whether the specified property has a value set, by inspecting
 * the boolean value of the has<PropertyName> property in propDescs.
 * If no such method is found for the property, a true value is returned.
 * @param obj
 * @param propDescs
 * @param property
 * @return True if the property of the specified Object parameter's specified
 *         PropertyDescriptor parameter has a value set.
 *
 */
static boolean hasValue( Object obj, HashMap propDescs, PropertyDescriptor property )
                            throws IllegalAccessException,
                                   InvocationTargetException {
  String hasPropertyName = "has" + capitalize( property.getName() );
  PropertyDescriptor hasPropertyDesc = (PropertyDescriptor) propDescs.get(hasPropertyName);
  if (hasPropertyDesc!=null) {
    Class hasPropertyType = hasPropertyDesc.getPropertyType();
    if (hasPropertyType==Boolean.TYPE) {
      return ((Boolean) readValue( hasPropertyDesc, obj )).booleanValue();
    }
  }
  return true;
}
```

```
  public static void p(String msg) {
    System.out.println(msg);
  }
}
```

13. FCRSchema.java

```
package mil.navy.nps.cs.oomi.fiom;

/**
 * @author LSC
 * @version 1.0
 */

public interface FCRSchema extends Schema {

}
```

14. FCRSemantics.java

```
package mil.navy.nps.cs.oomi.fiom;

/**
 * @author LSC
 * @version 1.0
 */

public interface FCRSemantics {

}
```

15. FCRSyntax.java

```
package mil.navy.nps.cs.oomi.fiom;

/**
 * @author LSC
 * @version 1.0
 */

public interface FCRSyntax {
}
```

16. FE.java

```
package mil.navy.nps.cs.oomi.fiom;

import java.net.URL;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;


/**
 * The OOMI Federation Entity.
 *
 * @author LSC
 * @version 1.0
 */

public interface FE {

  /**
   * Returns the FIOM that this FE belongs to.
   * @return Returns the FIOM that this FE belongs to.
   */
  public FIOM getFIOM();

  /**
   * Returns the name of this FE.
   */
  public String getFEName();

  /**
   * Returns the parent of this FE.
   * @return Returns the parent of this FE, returns null if this FE has no
   *         parent.
   */
  public FE getParent();


  /**
   * Returns the root FEV of this FE.
   * Automatically creates a new root FEV if one does not already exists.
   * @return Returns the root FEV of this FE.
   */
  public FEV getRootFEV();

  /**
   * Finds the FEV with the specified name.
   * @return Returns the FEV in this FE that has the specified name
   *         (not case-sensitive).
   *         Returns null if not found.
   */
  public FEV findFEV( String fevName );

  /**
   * Determines if there is a FEV with the specified name for this FE.
   */
  public boolean fevExists(String fevName);

  /**
   * Creates a new instance of FEV with the specified name,
   * replaces the current root FEV.
   */
  public FEV createRootFEV(String fevName);

  /**
   * Create a new child FE for this FE and returns the newly created child.
   * @return Returns the newly created child FE.
   */
  public FE newChild(String childFEName) throws DuplicateKeyException;



  /**
   * Returns the number of child FEs of this object.
   * @return Returns the number of child FEs of this object.
   */
  public int childCount();



  /**
   * Returns an array of child FEs of this FE.
   * @return Returns an array of child FEs of this FE, returns an empty array
   *         if there are no children.
   */
  public FE[] getChild();

  /**
   * Finds a descendent FE with the specified name.
   * @param descendentName The name of the descendent to find.
   * @param recurseTree    Whether to search in subtrees.
   * @return Returns a descendent FE with the specified name, returns null
   *         if not found.
   */
  public FE findDescendent(String descendentName, boolean recurseTree);



  /**
   * Finds and returns all CCRs within the hierarchy (including this FE)
   * using the specified SearchHandler.
   * @param handler The SearchHandler to use during search.
   */
  public void findCCR(SearchHandler handler);

  /**
   * Finds and returns all CCRs within the hierarchy
   * (only the descendents of this FE, excluding this FE)
   * using the specified SearchHandler.
   * @param handler The SearchHandler to use during search.
   */
  public void findDescendentCCR(SearchHandler handler);



  /**
   * Finds and returns the all CCRs within the hierarchy (including this FE)
   * with SystemName matching specified systemName.
   * @param systemName The system name to match.
   *
   * @return Returns an array of CCRs matching specified systemName,
   *          returns an empty array if no match found.
   */
  public CCR[] findCCR(String systemName);

  /**
   * Finds and returns the CCR within the hierarchy (including this FE)
   * with SystemName and XMLNameSpaceURI matching the
   * specified systemName and xmlNameSpaceURI.
```

```
   * @param systemName The system name to match.
   * @param xmlNameSpaceURI The XML schema URL to match.
   *
   * @return Returns the CCR matching the
   *           specified systemName and xmlNameSpaceURI,
   *           returns null if no match found.
   */
  public CCR findCCR(String systemName, URL xmlNameSpaceURI);

  /**
   * Finds and returns the CCR within the hierarchy (excluding this FE)
   * with SystemName and XMLNameSpaceURI matching the
   * specified systemName and xmlNameSpaceURI.
   * @param systemName The system name to match.
   * @param xmlNameSpaceURI The XML schema URL to match.
   *
   * @return Returns the CCR matching the
   *           specified systemName and xmlNameSpaceURI,
   *           returns null if no match found.
   */
  public CCR findDescendentCCR(String systemName, URL xmlNameSpaceURI);

}
```

17. FEV.java

```
package mil.navy.nps.cs.oomi.fiom;

import java.net.URL;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;



/**
 * The OOMI Federation Entity View.
 * @author LSC
 * @version 1.0
 */

public interface FEV {
```

```
/**
 * Returns the name of this FEV.
 */
public String getFEVName();

/**
 * Returns the FE that this FEV belongs to.
 */
public FE getFE();

/**
 * Returns the FCR corresponding to this FEV.
 * @return Returns the FCR corresponding to this FEV,
 *         if newFCR() had not been called before, returns null.
 */
public FCR getFCR();

/**
 * Returns the parent of this FEV.
 * @return Returns the parent of this FEV, returns null if this FEV has no
 *         parent.
 */
public FEV getParent();

/**
 * Creates a new instance of FCR for this FEV, replacing the existing FCR.
 * @return Returns the newly created instance.
 */
public FCR createFCR(String fcrName);

/**
 * Returns an array of child FEVs of this FEV.
 * @return Returns an array of child FEVs of this FEV, returns an empty array
 *         if there are no children.
 */
public FEV[] getChild();

/**
 * Returns the number of child FEVs of this object.
 * @return Returns the number of child FEVs of this object.
 */
public int childCount();
```

```
/**
 * Finds a descendent FEV with the specified name.
 * @param descendentName The name of the descendent to find.
 * @param recurseTree    Whether to search in subtrees.
 * @return Returns a descendent FEV with the specified name, returns null
 *         if not found.
 */
public FEV findDescendent(String descendentName, boolean recurseTree);

/**
 * Finds and returns the CCR within the hierarchy (including this FEV)
 * with SystemName and XMLNameSpaceURI matching the
 * specified systemName and xmlNameSpaceURI.
 * @param systemName The system name to match.
 * @param xmlNameSpaceURI The XML schema URL to match.
 *
 * @return Returns the CCR matching the
 *          specified systemName and xmlNameSpaceURI,
 *          returns null if no match found.
 */
public CCR findCCR(String systemName, URL xmlNameSpaceURI);

/**
 * Finds the CCRs within the hierarchy (including this FEV)
 * that satisfies the specified criteria.
 * @param criteria The criteria to match.
 */
public void findCCR(SearchHandler handler);

/**
 * Finds and returns the CCR within the sub-tree (excluding this FEV)
 * with SystemName and XMLNameSpaceURI matching the
 * specified systemName and xmlNameSpaceURI.
 * @param systemName The system name to match.
 * @param xmlNameSpaceURI The XML schema URL to match.
 *
 * @return Returns the CCR matching the
 *          specified systemName and xmlNameSpaceURI,
 *          returns null if no match found.
 */
public CCR findDescendentCCR(String systemName, URL xmlNameSpaceURI);
```

```
  /**
   * Create a new child FEV for this FEV and returns the newly created child.
   * @return Returns the newly created child FEV.
   */
  public FEV newChild(String childFEVName) throws DuplicateKeyException;

}
```

18. FIOM.java

```
package mil.navy.nps.cs.oomi.fiom;

import java.net.URL;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;



/**
 * Represents a OOMI Federation.
 * <p>
 * <ul>
 *
 * <li>Under a Federation, a CCR shall always be uniquely identified by the
 *     SystemName, CCRName pair.
 *
 * <li>The CCR can also be uniquely identified by the SystemName, XMLNameSpaceURI
 *     pair, if a XMLNameSpaceURI had been defined for the CCR.
 *
 * <li>For the Federation to be usable by the Translator, all CCRs expected to
 *     be received from any system through XML needs to have both
 *     the values of JavaClassName and XMLNameSpaceURI set appropriately.
 *
 * </ul>
 *
 * @author LSC
 * @version 1.0
 */

public interface FIOM {

  /**
 * Returns the name of this FIOM.
 *
 * @return Returns the name of this FIOM.
 */
public String getFIOMName() ;

/**
 * Returns the FIOMFactory instance for this FIOM object.
 */
public FIOMFactory factory();

/**
 * Returns all FEs defined in this FIOM as an array.
 * @return Returns an array of all FEs in this FIOM.
 *         A zero-length array is returned if there are no FEs in this FIOM.
 */
public FE[] getFE();

/**
 * Returns the number of FEs defined in this FIOM.
 * @return Returns the number of FEs defined in this FIOM.
 */
public int feCount();

/**
 * Creates a new FE instance and adds it to this FIOM.
 * @param feName The name of the new FE to be created.
 * @return Returns the newly created FE instance.
 */
public FE newFE(String feName) throws DuplicateKeyException;

/**
 * Finds and returns the FE with the specified name within this FIOM.
 * @param feName The name of the FE to find, a case-insensitive match
 *               is performed.
 * @return If there is an FE with the specified name is in this FIOM,
 *         the FE is returned, else null is returned.
 */
public FE findFE(String feName);


/**
 * Determines if a FE with the specified name exists in this FIOM.
 */
public boolean feExists(String feName);

/**
 * Finds and returns the CCR within this FIOM
 * with SystemName and XMLNameSpaceURI matching the
 * specified systemName and xmlNameSpaceURI.
 * @param systemName The system name to match.
 * @param xmlNameSpaceURI The XML namespace URI to match.
 *
 * @return Returns the CCR matching the
 *          specified systemName and xmlNameSpaceURI,
 *          returns null if no match found.
 */
public CCR findCCR(String systemName, URL xmlNameSpaceURI);



/**
 * Creates a new CCR within this FIOM with
 * systemName, ccrName, and xmlNameSpaceURI as
 * specified.  Provides independent CCR creation; CCR will be
 * associated with an FCR using joinFCR() method from class CCR.
 * @param systemName The system name to create CCR for.
 * @param ccrName The name of the new CCR being created.
 * @param xmlNameSpaceURI The XML namespace URI for the new CCR.
 *
 * @throws DuplicateKeyException when the specified systemName, ccrName pair
 *          or systemName, xmlNameSpaceURI pair already exist for this FIOM.
 */
public CCR newCCR(String systemName, String ccrName, URL xmlNameSpaceURI)
                                              throws DuplicateKeyException;

/**
 * Registers the specified CCR with the specified FCR,
 * removing the specified CCR from the unregistered list of this FIOM.
 * The specified CCR must be in the unregistered list of this FIOM,
 * else, nothing is done.
 */
public void registerCCR(FCR fcr, CCR ccr);

/**
 * Returns the number of unregistered CCRs in this FIOM.
 */
```

```
public int unregisteredCCRCount();

/**
 * Adds newly created CCR to the list of unregistered CCRs.
 * @param newCCR The newly created CCR.
 */
public void addUnregisteredCCR( CCR newCCR );

/**
 * Returns an array of all unregistered CCR contained in this FIOM.
 * @return Returns an array of all unregistered CCR contained in this FIOM.
 */
public CCR[] getUnregisteredCCR( );

/**
 * Searches list of unregistered CCRs to see if list contains CCR matching one
 * specified by systemName, ccrName pair.
 * @param systemName The name of the system the CCR was created for.
 * @param ccrName The name given to the newly created CCR.
 *
 * @return Returns the CCR matching the
 *   specified systemName and ccrName,
 *   returns null if no match found.
 */
public CCR findUnregisteredCCR(String systemName, String ccrName);

/**
 * Searches list of unregistered CCRs to see if list contains CCR matching one
 * specified by systemName, xmlNameSpaceURI pair.
 * @param systemName The name of the system the CCR was created for.
 * @param xmlNameSpaceURI The URL given to the newly created CCR.
 *
 * @return Returns the CCR matching the
 *   specified systemName and xmlNameSpaceURI,
 *   returns null if no match found.
 */
public CCR findUnregisteredCCR(String systemName, URL xmlNameSpaceURI);

public boolean ccrExists(String systemName, String ccrName);

public boolean ccrExists(String systemName, URL xmlNameSpaceURI);

public boolean equals(Object o);
```

```
/**
 * Adds a translation map to this FIOM.
 */
public void addTranslation( String fcrClassName, String ccrClassName,
                            String translationClassName );


/**
 * Finds the translation for the specified fcrClassName and ccrClassName.
 * @return Returns the fully qualified class name of the translation class.
 *         Returns null if no such translation exists.
 */
public String findTranslation( String fcrClassName, String ccrClassName );



/**
 * Creates new instances of TypeName.
 * @param typeName The type name (Java signature format).
 * @return Returns a new instance of TypeName representing the type specified by the
argument typeName.
 */
public TypeName makeTypeName( String typeName );
/**
 * Create a new instance of TypeName.
 * @param theClass The class of the type.
 * @return Returns a new instance of TypeName representing the class specified by the
argument theClass.
 */
public TypeName makeTypeName( Class theClass );
/**
 * Creates a new instance of TypeName that represents an array of the specified
 * type with the number of dimensions specified by dimensionCount.
 * @param typeName The type name (Java signature format) of the element of the array.
 * @param dimensionCount The number of dimensions the new array type should have.
 * @return Returns a new instance of TypeName representing
 *          an array with elements that are of the type specified
 *          by the argument typeName.
 */
public TypeName makeArrayTypeName( String typeName, int dimensionCount );
/**
 * Creates a new instance of TypeName that represents an array of the specified
 * type with the number of dimensions specified by dimensionCount.
 * @param typeName The class of the element of the array.
 * @param dimensionCount The number of dimensions the new array type should have.
```

```
   * @return Returns a new instance of TypeName representing
   *          an array with elements that are of the type specified
   *          by the argument theClass.
   */
  public TypeName makeArrayTypeName( Class theClass, int dimensionCount );



}
```

19. FIOMFactory.java

```
package mil.navy.nps.cs.oomi.fiom;

import java.lang.reflect.*;

/**
 * A class factory for all the interfaces defined in the
 * <code>mil.navy.nps.cs.oomi.fiom</code> package.
 * @author LSC
 * @version 1.0
 */

public class FIOMFactory {

  /**
   * The default factory, defaults to null.
   */
  static FIOMFactory _defaultFactory = null;

  Constructor _fiomConstructor;
  Constructor _ccrConstructor;

  Constructor _attributeConstructor;
  Constructor _attributeMappingConstructor;

  Constructor _typeNameConstructor;



  /**
   * @param attributeImplClass The implementation class of Attribute interface.
   */
  public FIOMFactory(
                    String fiomImplClass,
                    String ccrImplClass,
                    String attributeImplClass,
                    String attributeMappingImplClass,
                    String typeNameImplClass )
                      throws ClassNotFoundException,
                             NoSuchMethodException,
                             InvocationTargetException,
                             IllegalAccessException,
                             InstantiationException  {
  _fiomConstructor = getConstructor( Class.forName( fiomImplClass ),
                            new Class[] { String.class } );
  _ccrConstructor = getConstructor( Class.forName( ccrImplClass ),
                            new Class[] { FCR.class,
                                          String.class, String.class } );
  _attributeConstructor = getConstructor( Class.forName( attributeImplClass ),
                            new Class[] { Attribute.class,
                                          String.class, TypeName.class } );
  _attributeMappingConstructor = getConstructor( Class.forName( attributeMappingImplClass ),
                            new Class[] { Attribute[].class,
                                          Attribute.class } );
  _typeNameConstructor = getConstructor( Class.forName( typeNameImplClass ),
                            new Class[] {String.class} );
}

/**
 * Sets the default factory.
 */
public static void _setDefaultFactory(FIOMFactory f) {
  _defaultFactory = f;
}
public static FIOMFactory _defaultFactory() {
  return _defaultFactory;
}

protected Constructor getConstructor( Class cls, Class[] args )
                                  throws NoSuchMethodException {
  try {
    return cls.getConstructor(args);
  }
  catch (NoSuchMethodException exc) {
    throw constructorException( cls, args );
  }
```

```
}

protected NoSuchMethodException constructorException( Class cls, Class[] args ) {
  StringBuffer sb = new StringBuffer(50);
  if (args.length>0)
    sb.append(args[0].getName());
  for (int i=1; i<args.length;i++) {
    sb.append( ", " + args[i].getName() );
  }
  return new NoSuchMethodException( "public Constructor(" + sb.toString()
                                  + ") not found in " + cls.getName() );
}

/**
 * Creates new instances of Attributes.
 * The newly created Attribute will be added to the parent.
 */
public Attribute makeAttribute( Attribute parent,
                                String name, TypeName type ) {
  Object[] args = { parent, name, type };
  try {
    return (Attribute) _attributeConstructor.newInstance(args);
  }
  catch (Exception exc) {
    throw new RuntimeException(exc.getMessage());
  }
}

protected static void p1(String m) {
  System.out.println(m);
}

/**
 * Creates new instance of CCR.
 */
public CCR makeCCR( FCR fcr, String systemName, String ccrName ) {
  Object[] args = { fcr, systemName, ccrName };
  return (CCR) newInstance( _ccrConstructor, args );
}

/**
 * Creates new instances of TypeName.
 * @param typeName The type name (Java signature format).
```

```
   * @return Returns a new instance of TypeName representing the type specified by the
argument typeName.
   */
  public TypeName makeTypeName( String typeName ) {
    Object[] args = { typeName };
    return (TypeName) newInstance( _typeNameConstructor, args );
  }
  /**
   * Create a new instance of TypeName.
   * @param theClass The class of the type.
   * @return Returns a new instance of TypeName representing the class specified by the
argument theClass.
   */
  public TypeName makeTypeName( Class theClass ) {
    return makeTypeName(theClass.getName());
  }
  /**
   * Creates a new instance of TypeName that represents an array of the specified
   * type with the number of dimensions specified by dimensionCount.
   * @param typeName The type name (Java signature format) of the element of the array.
   * @param dimensionCount The number of dimensions the new array type should have.
   * @return Returns a new instance of TypeName representing
   *          an array with elements that are of the type specified
   *          by the argument typeName.
   */
  public TypeName makeArrayTypeName( String typeName, int dimensionCount ) {
    Object[] args = { encodeTypeName(typeName,dimensionCount) };
    return (TypeName) newInstance( _typeNameConstructor, args );
  }
  /**
   * Creates a new instance of TypeName that represents an array of the specified
   * type with the number of dimensions specified by dimensionCount.
   * @param typeName The class of the element of the array.
   * @param dimensionCount The number of dimensions the new array type should have.
   * @return Returns a new instance of TypeName representing
   *          an array with elements that are of the type specified
   *          by the argument theClass.
   */
  public TypeName makeArrayTypeName( Class theClass, int dimensionCount ) {
    return makeArrayTypeName( theClass.getName(), dimensionCount );
  }

  /**
   * Creates an instance of AttributeMapping.
```

```
 */
public AttributeMapping makeAttributeMapping(
                            Attribute[] fromAttributes,
                            Attribute   toAttribute ) {
  Object[] args = { fromAttributes, toAttribute };
  return (AttributeMapping) newInstance( _attributeMappingConstructor, args );
}


/**
 * Creates new instances of the class declaring the specified constructor.
 */
protected Object newInstance( Constructor constructor, Object[] args ) {
  try {
    return constructor.newInstance(args);
  }
  catch (Exception exc) {
    throw new RuntimeException(exc.getClass().getName() + ": " + exc.getMessage());
  }
}


/**
 * Decodes the type (in internal Java signature format)
 * into source code format (human-friendly format).
 */
public static String decodeTypeName(String qualifiedTypeName) {
  if (qualifiedTypeName.indexOf('[')!=-1) {
    StringBuffer result = new StringBuffer();
    String className = qualifiedTypeName;
    int len = className.length();
    int i = 0;
    while (className.charAt(i)=='[') {
      result.append("[]");
      i++;
    }
    String type;
    switch (className.charAt(i)) {
      case 'B': type = byte.class.getName(); break;
      case 'C': type = char.class.getName(); break;
      case 'D': type = double.class.getName(); break;
      case 'F': type = float.class.getName(); break;
      case 'I': type = int.class.getName(); break;
      case 'J': type = long.class.getName(); break;
      case 'S': type = short.class.getName(); break;
      case 'Z': type = boolean.class.getName(); break;
      case 'L': type = className.substring(i+1,len-1); break;
      default:
        type = "";
    }
    result.insert(0, type);
    return result.toString();
  }
  else
    return qualifiedTypeName;
}


/**
 * Encodes the specified type name (non-array type)
 * into its qualified type name (internal Java signature format)
 * as an array.
 *
 * @param qualifiedTypeName The non-array qualified type name to be the
 *                          element type of the array.
 * @param dimCount The number of dimensions the array should have.
 *
 * @return The internal Java signature of array,
 *          if the parameter dimCount==0, parameter qualifiedTypeName is
 *          returned with no change.
 */
public static String encodeTypeName(String qualifiedTypeName,
                                            int dimCount) {
  String result;
  if (dimCount==0)
    return qualifiedTypeName;
  // encoding needed only if dimension is 1 or more
  if (qualifiedTypeName.equals(byte.class.getName()))
    result = "B";
  else if (qualifiedTypeName.equals(char.class.getName()))
    result = "C";
  else if (qualifiedTypeName.equals(double.class.getName()))
    result = "D";
  else if (qualifiedTypeName.equals(float.class.getName()))
    result = "F";
  else if (qualifiedTypeName.equals(int.class.getName()))
```

```
      result = "I";
    else if (qualifiedTypeName.equals(long.class.getName()))
      result = "J";
    else if (qualifiedTypeName.equals(short.class.getName()))
      result = "S";
    else if (qualifiedTypeName.equals(boolean.class.getName()))
      result = "Z";
    else
      result = "L" + qualifiedTypeName + ";";
    StringBuffer sb = new StringBuffer(dimCount);
    for (int i=0; i<dimCount; i++) {
      sb.append("[");
    }
    return sb + result;
  }

}
```

20. Schema.java

```
package mil.navy.nps.cs.oomi.fiom;

import java.util.List;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

/**
 * @author LSC
 * @version 1.0
 */

public interface Schema {

  /**
   * Returns actual type name of the class
   * this Schema object describes.
   */
  public TypeName getType();
  /**
   * Sets actual type name of the class
   * this Schema object describes.
   */
  public void   setType(TypeName type);

  /**
   * Adds a new attribute to this schema.
   */
  public void addAttribute( Attribute attr );



  /**
   * Adds a top level Attribute to this schema.
   * @param parent The parent of the new attribute.
   * @param name The name of the new attribute.
   * @param type The type name of the new attribute.
   * @return A reference to the newly added attribute.
   */
  public Attribute addAttribute( String name, TypeName type );

  /**
   * Removes the specified attribute from the top-level list.
   */
  public void removeAttribute( Attribute attr );

  /**
   * Returns an array of top-level attributes.
   */
  public Attribute[] getAttribute();

  /**
   * Returns the attribute in this Schema
   * that is equal to the specified Attribute.
   */
  public Attribute findAttribute( Attribute attr );

  /**
   * Enumerates all attributes and their descendents and add them to the specified list,
   * effectively flattening the attribute tree into a list.
   */
  public void enumerateAttributes( List result );
  /**
   * Enumerates all attributes and their descendents and puts them in array
   * and returns this array.
   */
```

```
  public Attribute[] enumerateAttributes(  );

}
```

21. TypeName.java

```
package mil.navy.nps.cs.oomi.fiom;

/**
 * Represents a type name in the internal Java signature format.
 * This is introduced to prevent confusion when using only a String type
 * to specify both Name and Type of an attribute.
 * The type name represented by an instance of TypeName does not necessarily
 * have to be valid at runtime, that is, the type name can be a non-compiled
 * type.
 *
 * @author LSC
 * @version 1.0
 */

public interface TypeName {

  /**
   * An array of names of primitive types.
   */
  final String[] PRIMITIVE_NAMES = {
                          boolean.class.getName(),
                          byte.class.getName(),
                          char.class.getName(),
                          short.class.getName(),
                          int.class.getName(),
                          long.class.getName(),
                          float.class.getName(),
                          double.class.getName()
                         };



  /**
   * Returns the type name in internal Java signature format.
   */
  public String toString();

  /**
```

```
   * Duplicates this TypeName object.
   */
  public TypeName copy();

  /**
   * Returns if the specified obj is of the same type,
   * true if and only if  obj implements TypeName
   *                      and this.toString().equals( obj.toString() )
   */
  public boolean equals(Object obj);

  /**
   * Implementor shall ensure that for any TypeName t1, t2,
   * t1.equals(t2) ==> t1.hashCode()==t2.hashCode().
   */
  public int hashCode();

  /**
   * Indicates if the type represented by this TypeName object is an array.
   */
  public boolean isArray();

  /**
   * Indicates if the type represented by this TypeName object is a primitive type.
   */
  public boolean isPrimitive();

}
```

C. PACKAGE: mil.navy.nps.cs.oomi.exceptions

1. DuplicateKeyException.java

```
package mil.navy.nps.cs.oomi.exceptions;

/**
 * Indicates a duplicate FIOM name was found.
 *
 * @author LSC
 * @version 1.0
 */

public class DuplicateKeyException extends OOMIException {
  public DuplicateKeyException(String msg) {
    super(ErrorCodes.DUPLICATE_KEY, msg);
  }
}
```

2. ErrorCodes.java

```
package mil.navy.nps.cs.oomi.exceptions;

/**
 * Defines the set of error codes.
 * @author LSC
 * @version 1.0
 */

public class ErrorCodes {

  /**
   * Not to be instantiated.
   */
  private ErrorCodes() {
  }

  /**
   * Default error code value.
   */
  public static final int UNKNOWN = -1;

  /**
   * Indicates a duplicate key value encountered.
   */
  public static final int DUPLICATE_KEY = -10;

  /**
   * Indicates a unique name violation.
   */
  public static final int UNIQUE_NAME_VIOLATION = -20;

  /**
   * Indicates an unknown URI.
   */
  public static final int UNKNOWN_URI = -30;
  /**
   * Indicates an unmarshalling exception.
   */
  public static final int UNMARSHAL = -200;
  /**
   * Indicates an unmarshalling exception.
   */
  public static final int MARSHAL = -210;
}
```

3. IntrospectionException.java

```
package mil.navy.nps.cs.oomi.exceptions;

/**
 *
 * @author LSC
 * @version 1.0
 */

public class IntrospectionException extends Exception {

  public IntrospectionException(String s) {
    super(s);
  }
  public IntrospectionException(Exception e) {
    super(e.getMessage());
  }
}
```

4. OOMIException.java

```
package mil.navy.nps.cs.oomi.exceptions;

/**
 * @author LSC
 * @version 1.0
 */

public class OOMIException extends Exception {

  /**
   * The error code of this exception.
   */
  int _errorCode = ErrorCodes.UNKNOWN;

  public OOMIException( int errorCode, String errorMsg ) {
    super(errorMsg);
    _errorCode = errorCode;
  }

}
```

5. UniqueNameViolationException.java

```
package mil.navy.nps.cs.oomi.exceptions;

/**
 * @author LSC
 * @version 1.0
 */

public class UniqueNameViolationException extends OOMIException {
  public UniqueNameViolationException(String msg) {
    super(ErrorCodes.UNIQUE_NAME_VIOLATION, msg);
  }
}
```

6. UnknownNameSpaceURIException.java

```
package mil.navy.nps.cs.oomi.exceptions;

import java.net.URL;

/**
 * @author LSC
 * @version 1.0
 */

public class UnknownNameSpaceURIException extends OOMIException {

  public UnknownNameSpaceURIException(URL unknownURI) {
    super( ErrorCodes.UNKNOWN_URI,
           "The namespace URI: " + unknownURI +
            " is not defined in this Federation." );
  }
}
```

7. UnmarshalException.java

```
package mil.navy.nps.cs.oomi.exceptions;

/**
 * @author LSC
 * @version 1.0
 */

public class UnmarshalException extends OOMIException {

  public UnmarshalException(String msg) {
    super (ErrorCodes.UNMARSHAL, msg);
  }
}
```

D. PACKAGE: mil.navy.nps.cs.oomi.impl

1. AttributeImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.List;
import java.util.Arrays;
import java.util.Iterator;
import java.util.ListIterator;
import java.util.ArrayList;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

/**
 * Directly manipulates the _children field
 *  of mil.navy.nps.cs.oomi.translator.Attribute.
 * <p>
 * May be modified to inherit from mil.navy.nps.cs.oomi.translator.Attribute since this class
 * might be used for code generation of FCRs in future.
 *
 * <p>
 *
 * @author LSC
 * @version 1.0
 */

public class AttributeImpl implements mil.navy.nps.cs.oomi.fiom.Attribute {

  /**
   * The parent of this attribute.
   */
  Attribute _parent = null; // defaults to null, indicating no parent

  /**
   * All the children that composes this attribute.
   *
   */
  protected List _childList = new ArrayList();

  /**
   * Attribute name.
   */
  protected String _name = null;

  /**
   * Attribute type.
   */
  protected TypeName _type = null;


    //-
   //-
  //-
  /**
   * MinOccurs.
   */
  protected int _minOccurs = 0;
  /**
   * MaxOccurs.
   */
  protected int _maxOccurs = 0;

  /**
   * Public no-argument constructor required for Castor Unmarshaller.
   */
  public AttributeImpl() {
  }
```

```
/**
 * Creates an instance with specified parent, name and type.
 */
public AttributeImpl( Attribute parent, String name, TypeName type ) {
  //setParent(parent);
  setName(name);
  setType(type);
  if (parent==null)
    setParent(null);
  else
    ((AttributeImpl)parent).addChild(this);

}

/**
 * Sets the parent.
 */
protected void setParent(Attribute val) {
  this._parent = val;
}
public Attribute getParent() {
  return this._parent;
}
public void setName (String name) {
  _name = name;
}

public void setType (TypeName type) {
  _type = type;
}

public String getName() {
  return _name ;
}
public TypeName getType() {
  return _type;
}

/**
 * Performs a shallow copy from s to d,
 * less the _type, _name and _childList field.
 */
```

```
  protected static void copyContent( AttributeImpl s, AttributeImpl d ) {
    d.setMaxOccurs(s.getMaxOccurs());
    d.setMinOccurs(s.getMinOccurs());
  }

  /**
   * Clones the descendents of the specified AttributeImpl object and
   * make this object the parent of the clones.
   */
  protected void cloneDescendents( AttributeImpl attr ) {
    //p( "cloneDesc: " + attr.fullPath() + " " + attr.childCount() );
    if (attr.childCount()==0)
      return;
    else {
      Iterator iter = attr._childList.iterator();
      while (iter.hasNext()) {
        AttributeImpl child = (AttributeImpl) iter.next();
        AttributeImpl childCopy;
        try {
          childCopy = (AttributeImpl) newChild( child.getName(), child.getType().copy() );
          copyContent( child, childCopy );
          childCopy.cloneDescendents( child );
        }
        catch (UniqueNameViolationException exc) {
          // ignore this exception, since it should never occur when cloning
          // an attribute, in case it occurs, throw a RuntimeException
          throw     new     RuntimeException(     "cloneDescendents(Attribute)     encountered
UniqueNameViolationException" );
        }
      }
    }
  }

  public Attribute copy() {
    FIOMFactory f = FactoryImpl.factory();
    if (getParent()==null) {
      AttributeImpl    result    =    (AttributeImpl)    f.makeAttribute(null,    getName(),
getType().copy());
      copyContent( this, result );
      result.cloneDescendents(this);
      return result;
    }
    else {
      // Copy the ancestors
      Attribute[] array = pathToArray();
      Attribute parent = null;
      AttributeImpl result = null;
      for (int i=0; i<array.length; i++) {
        result     =     (AttributeImpl)     f.makeAttribute(parent,     array[i].getName(),
array[i].getType().copy());
        copyContent( this, result );
        parent = result;
      }
      result.cloneDescendents(this);
      return result;
    }
  }



  public int childCount() {
    return _childList.size();
  };

  public Attribute[] getChild() {
    return (Attribute[]) _childList.toArray( new Attribute[childCount()] );
  }
  /**
   * Adds the specified child attribute to this AttributeImpl.
   */
  public void addChild(AttributeImpl attr) {
    _childList.add(attr);
    attr.setParent(this);
  }
  /**
   * Sets the list of children with the specified array.
   * Generally, to be called only from the Castor Unmarshaller.
   */
  public void setChild( Attribute[] v ) {
    _childList.clear();
    for (int i=0; i<v.length; i++) {
      this.addChild((AttributeImpl) v[i]);
    }
  }
  public Attribute findChild(String attributeName) {
    Iterator iter = _childList.iterator();
    Attribute curr;
    while (iter.hasNext()) {
```

```
    curr = (Attribute) iter.next();
    if (curr.getName().equalsIgnoreCase(attributeName))
      return curr;
  }
  return null;
}

public Attribute findDescendent(Attribute attr) {
  Iterator iter = _childList.iterator();
  Attribute curr;
  Attribute c;
  while (iter.hasNext()) {
    curr = (Attribute) iter.next();
    if (curr.equals(attr))
      return curr;
    c = curr.findDescendent(attr);
    if (c!=null)
      return c;
  }
  return null;
}

public boolean removeDescendent( Attribute descendentToRemove ) {
  // Assumption: any Attribute does not appear more than once in a hierarchy.
  ListIterator iter = _childList.listIterator();
  boolean done = false;
  Object curr;
  while (iter.hasNext() && !done) {
    curr = iter.next();
    if (curr==descendentToRemove) {
      iter.remove();
      ((AttributeImpl)curr).setParent(null);
      done = true;
    }
  }
  if (!done) {
    Iterator i = _childList.iterator();
    while (i.hasNext() && !done) {
      done = ((Attribute)(i.next())).removeDescendent(descendentToRemove);
    }
  }
  return done;
}
```

```
  public Attribute newChild( String attributeName, TypeName attributeType)
                                throws UniqueNameViolationException {
    if ( findChild(attributeName) != null )
      throw new UniqueNameViolationException(
                          "Attribute " +
                          getName() +
                          " already has a child attribute named " +
                          attributeName );
    Attribute    result    =       FactoryImpl.factory().makeAttribute(this,    attributeName,
attributeType);
                        //new AttributeImpl(this, attributeName, attributeType);
    //_childList.add(result);
    return result;
  }

  public int hashCode() {
    return this.fullPath().hashCode();
  }



  public boolean equals( Object o ) {
    if (this==o)
      return true;
    Attribute attr = (Attribute) o;
    if ( getName()==null || getType()==null ||
         attr.getName()==null || attr.getType()==null )
      return false;
    if ( getName().equals(attr.getName()) && getType().equals(attr.getType()) ) {
      if ( getParent()==attr.getParent() )
        return true;
      else if ( getParent()==null || attr.getParent()==null )
        return false;
      else
        return getParent().equals(attr.getParent());
    }
    else
      return false;
  }

  public void enumerateDescendents( List result ) {
    Iterator i = _childList.iterator();
    Attribute attr;
    while (i.hasNext()) {
      attr = (Attribute) i.next();
      result.add(attr);
      attr.enumerateDescendents(result);
    }
  }

  public boolean isMandatory() {
    return getMinOccurs()>=1;
  }

  public boolean isArray() {
    return _type.isArray();
    //return getMaxOccurs()>1;
  }

  public String fullPath() {
    List path = new ArrayList(5);
    Attribute curr = this;
    while (curr!=null) {
      path.add(curr);
      curr = curr.getParent();
    }
    StringBuffer result = new StringBuffer();
    Object[] objs = path.toArray();
    Attribute attr;
    int start = objs.length-1;
    attr = (Attribute) objs[start];
    result.append(attr.getName());
    for (int i=start-1; i>=0; i--) {
      attr = (Attribute) objs[i];
      result.append(Attribute.NAME_SEPARATOR + attr.getName());
    }
    return result.toString() ;
  }

  public Attribute[] pathToArray() {
    List path = new ArrayList(5);
    Attribute curr = this;
    while (curr!=null) {
      path.add(curr);
      curr = curr.getParent();
    }
```

```
    Attribute[] result = new Attribute[path.size()];
    // reverse the order
    if (result.length>0) {
      int i = result.length-1;
      Iterator iter = path.iterator();
      Attribute a;
      while (iter.hasNext()) {
        a = (Attribute)iter.next();
        result[i] = a;
        i--;
      }
    }
    return result;
  }

    //-
   //-    Getters and Setters.
  //-
  public int getMinOccurs() {
    return _minOccurs;
  }
  public void setMinOccurs( int v ) {
    _minOccurs = v;
  }
  public int getMaxOccurs() {
    return _maxOccurs;
  }
  public void setMaxOccurs( int v ) {
    _maxOccurs = v;
  }
  public static void p(String m) {
    System.out.println(m);
  }
}
```

2. AttributeMappingImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import mil.navy.nps.cs.oomi.fiom.*;

public class AttributeMappingImpl implements AttributeMapping {
```

```
/**
 * Source attributes.
 */
protected Attribute _sourceAttr[];

/**
 * Target attribute.
 */
protected Attribute _targetAttr;

/**
 * Holds the many-to-one mapping from attributes of
 * one class to the attribute of another class.
 *
 */
public AttributeMappingImpl( Attribute source[], Attribute target ) {
  _sourceAttr = new Attribute[source.length];
  System.arraycopy( source, 0, _sourceAttr, 0, source.length );
  _targetAttr = target;
}

/**
 * Returns a copy of the array of from attributes.
 */
public Attribute[] getSourceAttributes() {
  Attribute[] result = new Attribute[_sourceAttr.length];
  System.arraycopy(_sourceAttr, 0, result, 0, _sourceAttr.length);
  return result;
}

public String toString() {
  if (_sourceAttr.length>0) {
    StringBuffer result = new StringBuffer(64);
    result.append( _sourceAttr[0].fullPath() );
    for (int i=1; i<_sourceAttr.length; i++) {
      result.append( ", " );
      result.append( _sourceAttr[i].fullPath() );
    }
    result.append(" --> " + _targetAttr.fullPath());
    return result.toString();
  }
  else
```

```
      return "";
  }

  /**
   * Returns the target attribute.
   */
  public Attribute getTargetAttribute() {
    return _targetAttr;
  }

  public Attribute[] fromAttributes() {
    return getSourceAttributes();
  }

  public Attribute toAttribute() {
    return getTargetAttribute();
  }

}
```

3. CCRImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.*;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;


/**
 * A reference implementation of the Component Class Representation (CCR).
 * Needs to maintain the mandatory attributes of the corresponding FCR.
 *
 * @author LSC
 * @version 1.0
 */

public class CCRImpl implements CCR, Comparable {

  /**
   * FCR to CCR attribute mappings.
   */
  protected List _fcrToCcrAttrMappingList = new ArrayList();

  /**
   * The system name of this CCR.
   */
  protected String _systemName = "";

  /**
   * The name of this CCR.
   */
  protected String _ccrName = "";

  /**
   * The FCR corresponding to this CCR.
   */
  protected FCR _fcr = null;

  /**
   * The schema for this CCR.
   */
  protected CCRSchema _ccrSchema = new CCRSchemaImpl();

  /**
   * The fully qualified name of the Castor generated class for the XML schema.
   */
  protected String _javaClassName = "";

  /**
   * The URL to the XML schema for which this CCR is defined for.
   */
  protected String _xmlNameSpaceURI = "";

  /**
   * An array of mandatory FCR attributes.
   * Transient field, built from _fcrToCcrAttrMappingList when required.
   */
  protected Set _mandatoryFCRAttributes = new HashSet();

  /**
   * Constructor required by Castor Unmarshaller.
   */
  public CCRImpl() {
  }

  /**
   * Initializes the CCR.
   * If the specified FCR is not null, this CCR shall join the specified FCR.
   * @param fcr         The FCR this CCR will join.
   * @param systemName  System name of this CCR.
   * @param ccrName     Name of this CCR.
   */
  public CCRImpl( FCR fcr, String systemName, String ccrName ) {
    joinFCR(fcr);
    setCCRName(ccrName);
    setSystemName(systemName);
  }

  /**
   * Sets the name of this CCR.
   * Public access required by Castor Unmarshaller.
   */
  public void setCCRName(String ccrName) {
    _ccrName = ccrName;
  }

  public String getCCRName() {
    return _ccrName;
  }

  /**
   * Sets the FCR of this object.
   * Used only for Castor Marshaling/Unmarshaling.
   */
  void setFCR(FCR fcr) {
    _fcr = fcr;
  }

  public FCR getFCR() {
    return _fcr;
  }

  public void joinFCR(FCR newFCR) {
    if (newFCR!=null) {
      ((FCRImpl) newFCR).addCCR( this );
    }
```

```
}

public CCRSchema getCCRSchema() {
  return _ccrSchema;
}
/**
 * Public access required by Castor Unmarshaller.
 */
public void setCCRSchema( CCRSchema v ) {
  _ccrSchema = v;
}

public String getJavaClassName() {
  return _javaClassName;
}

public void setJavaClassName( String javaClassName ) {
  _javaClassName = javaClassName;
}

public String getXMLNameSpaceURI() {
  return _xmlNameSpaceURI;
}

public void setXMLNameSpaceURI( String xmlNameSpaceURI ) {
  _xmlNameSpaceURI = xmlNameSpaceURI;
}

public String getSystemName() {
  return _systemName;
}
public void setSystemName(String v) {
  _systemName = v;
}
public AttributeMapping[] getFCRToCCRAttributeMapping() {
  return (AttributeMapping[]) _fcrToCcrAttrMappingList.toArray(
                              new AttributeMapping[_fcrToCcrAttrMappingList.size()] );
}
public void setFCRToCCRAttributeMapping( AttributeMapping[] v ) {
  clearFCRToCCRAttributeMappings();
  for (int i=0; i<v.length; i++) {
    _fcrToCcrAttrMappingList.add(v[i]);
  }
```

```
}
public Attribute[] mandatoryFCRAttributes() {
  return (Attribute[]) mandatoryFCRAttributeSet().toArray(
                        new Attribute[mandatoryFCRAttributeCount()] );
}
public boolean isMandatoryFCRAttribute( Attribute fcrAttr ) {
  return mandatoryFCRAttributeSet().contains(fcrAttr);
}
public int mandatoryFCRAttributeCount() {
  return mandatoryFCRAttributeSet().size();
}

/**
 * Returns the mandatory FCR attributes set, builds the set by calling
 * {@link #buildMandatoryFCRAttributeSet()} if the set is empty.
 */
protected Set mandatoryFCRAttributeSet() {
  if (_mandatoryFCRAttributes.size()==0)
    buildMandatoryFCRAttributeSet();
  return _mandatoryFCRAttributes;
}

/**
 * Builds the mandatory FCR attributes set, based on the current
 * _fcrToCcrAttrMappingList.
 */
protected void buildMandatoryFCRAttributeSet() {
  // first, clear the existing set
  _mandatoryFCRAttributes.clear();
  CCRSchema ccrSchema = getCCRSchema();
  Iterator iter = _fcrToCcrAttrMappingList.iterator();
  while (iter.hasNext()) {
    AttributeMapping mapping = (AttributeMapping)iter.next();
    Attribute[] fcrAttrs = mapping.fromAttributes();
    Attribute   ccrAttr  = mapping.toAttribute();
    if ( ccrSchema.findAttribute(ccrAttr).isMandatory() ) {
      for (int i=0; i<fcrAttrs.length; i++) {
        _mandatoryFCRAttributes.add(fcrAttrs[i]);
      }
    }
  }
}
```

```
  public void addFCRToCCRAttributeMapping( Attribute[] from, Attribute to ) {
    AttributeMapping mapping = FactoryImpl.factory().makeAttributeMapping( from, to );
    _fcrToCcrAttrMappingList.add( mapping );
  }

  public void clearFCRToCCRAttributeMappings() {
    _fcrToCcrAttrMappingList.clear();
  }

  /**
   * Used mainly for sorting required during testing.
   * Ordering is based on SystemName order followed by CCRName.
   */
  public int compareTo(Object o) {
    CCR ccr = (CCR) o;
    if (this==o)
      return 0;
    int c;
    c = getSystemName().compareTo(ccr.getSystemName());
    if (c!=0)
      return c;
    else
      return getCCRName().compareTo(ccr.getCCRName());
  }
  static void p(String m) {
    System.out.println(m);
  }
}
```

4. CCRSchemaImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

/**
 * @author LSC
 * @version 1.0
 */

public class CCRSchemaImpl extends SchemaImpl
```

```
                           implements CCRSchema {

  /**
   * Public no-arg constructor required by Castor Unmarshaller.
   */
  public CCRSchemaImpl() {
  }



}
```

5. FactoryImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import mil.navy.nps.cs.oomi.fiom.*;

/**
 * @author LSC
 * @version 1.0
 */

public class FactoryImpl {

  static FIOMFactory _defaultFactory = null;

  protected FactoryImpl() {
  }

  public static FIOMFactory newFIOMFactory() {
    try {
      FIOMFactory result
        = new FIOMFactory(
                           FIOMImpl.class.getName(),
                           CCRImpl.class.getName(),
                           AttributeImpl.class.getName(),
                           AttributeMappingImpl.class.getName(),
                           TypeNameImpl.class.getName() );
      return result;
    }
    catch (Exception exc) {
      throw new RuntimeException(exc.getClass() + ": " + exc.getMessage());
    }
  }

  public static FIOMFactory factory() {
    if (_defaultFactory==null)
      _defaultFactory = newFIOMFactory();
    return _defaultFactory;
  }

}
```

6. FCRImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.HashSet;
import java.util.Arrays;
import java.util.Collections;
import java.util.Collection;
import java.net.URL;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;


/**
 *
 * @author LSC
 * @version 1.0
 */

public class FCRImpl implements FCR {

  /**
   * The name of this FCR.
   */
  protected String _fcrName = "";

  /**
   * The Java class name that implements the FCRInstance for this FCR.
   */
  protected String _javaClassName;

  /**
   * The FEV that corresponds to this FCR.
   */
  protected FEV _fev = null;

  /**
   * The list of CCRs corresponding to this FCR.
   */
  protected Set _ccrSet = new HashSet();

  /**
   * The schema for this FCR.
   */
  protected FCRSchema _fcrSchema = new FCRSchemaImpl();

  /**
   * Required for Castor Marshaller and Unmarshaller.
   */
  public FCRImpl() {

  }

  /**
   * Creates a new instance with specified name.
   */
  public FCRImpl(FEV fev, String fcrName) {
    setFCRName(fcrName);
    setFEV(fev);
  }

  public String getFCRName() {
    return _fcrName;
  }

  /**
   * Sets the name of this FCR.
   * To be used only by the constructor
   * and during restoration from persistent storage, using
   * Castor Unmarshaller.
   */
  public void setFCRName(String fcrName) {
    _fcrName = fcrName;
  }

  /**
   * Sets the FEV of this FCR.
   * To be used only by the constructor
   * and by Castor Unmarshaller during restoration from persistent storage.
   *
   */
  public void setFEV(FEV value) {
    _fev = value;
  }



  public CCR[] getCCR() {
    return (CCR[])_ccrSet.toArray(new CCR[ccrCount()]);
  }

  public FEV getFEV() {
    return _fev;
  }

  public int ccrCount() {
    return _ccrSet.size();
  }

  public void findCCR(SearchHandler handler) {
    Iterator iter = _ccrSet.iterator();
    CCR ccr;
    while (iter.hasNext()) {
      ccr = (CCR)iter.next();
      if ( handler.satisfied(ccr) )
        handler.add(ccr);
    }
  }

  public CCR[] findCCR( String systemName ) {
    java.util.HashSet result = new java.util.HashSet();
    findCCR( makeCCRSystemNameSearchHandler(systemName, result) );
    return (CCR[]) result.toArray(new CCR[result.size()]);
    /*
    List result = new ArrayList();
    Iterator iter = _ccrList.iterator();
    CCR ccr;
    while (iter.hasNext()) {
      ccr = (CCR)iter.next();
      if ( ccr.getSystemName().equalsIgnoreCase(systemName) )
        result.add(ccr);
    }
    return (CCR[]) result.toArray(new CCR[result.size()]);
    */
  }
  public CCR findCCR(String systemName, String ccrName) {
    Iterator iter = _ccrSet.iterator();
    CCR ccr;
    while (iter.hasNext()) {
      ccr = (CCR)iter.next();
      if ( ccr.getSystemName().equalsIgnoreCase(systemName) &&
           ccr.getCCRName().equalsIgnoreCase(ccrName) )
        return ccr;
    }
    return null;
  }
  public CCR findCCR(String systemName, URL xmlNameSpaceURI) {
    if (xmlNameSpaceURI==null)
      return null;
    Iterator iter = _ccrSet.iterator();
    CCR ccr;
    String urlString = xmlNameSpaceURI.toString();
    while (iter.hasNext()) {
      ccr = (CCR)iter.next();
      if ( ccr.getSystemName().equalsIgnoreCase(systemName) &&
           ccr.getXMLNameSpaceURI().equals(urlString) )
        return ccr;
    }
    return null;
  }

  /*
  public CCR findCCR( URL xmlNameSpaceURI ) {
    if (xmlNameSpaceURI==null)
      return null;
    Iterator iter = _ccrList.iterator();
    CCR ccr;
```

```
    String urlString = xmlNameSpaceURI.toString();
    while (iter.hasNext()) {
      ccr = (CCR)iter.next();
      if ( ccr.getXMLNameSpaceURI().equals(urlString) )
        return ccr;
    }
    return null;
  }
  */
  public boolean ccrExists(String systemName, String ccrName) {
    return findCCR(systemName, ccrName)!=null;
  }
  public boolean ccrExists(String systemName, URL xmlNameSpaceURI) {
    return findCCR(systemName, xmlNameSpaceURI)!=null;
  }



  public CCR newCCR(String systemName, String ccrName, URL xmlNameSpaceURI)
                                              throws DuplicateKeyException {
    if ( ccrExists(systemName, ccrName) )
      throw new DuplicateKeyException(ccrName + " already exists for system: " + systemName);
    if ( ccrExists(systemName, xmlNameSpaceURI) )
      throw  new  DuplicateKeyException(xmlNameSpaceURI  +  "  already  exists  for  system:  "  +
systemName);
    CCRImpl ccr = new CCRImpl(this, systemName, ccrName);
    if ( xmlNameSpaceURI!=null)
      ccr.setXMLNameSpaceURI( xmlNameSpaceURI.toString() );
    _ccrSet.add(ccr);
    return ccr;
  }

  /**
   * Adds the specified CCR object to this FCR,
   * <strong> the FCR field of the CCRImpl parameter is updated to
   * point to this object. </strong>
   * <p>Also used by Castor Unmarshaller only.
   *
   */
  public void addCCR( CCR newCCR ) {
    ((CCRImpl)newCCR).setFCR(this);
    _ccrSet.add(newCCR);
  }
```

```
public FCRSchema getFCRSchema() {
  return _fcrSchema;
}
/**
 * Public access required by Castor Unmarshaller.
 */
public void setFCRSchema( FCRSchema v ) {
  _fcrSchema = v;
}

public String getJavaClassName() {
  return _javaClassName;
}
public void setJavaClassName(String className) {
  _javaClassName = className;
}

/**
 * A factory method that returns an instance of SearchHandler for search of
 * CCRs by system name of the CCRs.
 */
public static SearchHandler makeCCRSystemNameSearchHandler(
                              String systemName,
                              java.util.Set resultSet) {
  return new CCRSystemNameSearchHandler( systemName, resultSet );
}

/**
 * @param systemName
 * @param xmlNamespaceURI
 * @param result A single element array to hold the result, the element holds
 *               a null value if no match found.
 */
public static CCRKeySearchHandler makeCCRKeySearchHandler(
                                    String systemName, URL xmlNamespaceURI,
                                    CCR[] result) {
  if (result.length<1)
    throw new IllegalArgumentException( "Argument result must not be an empty array." );
  return new CCRKeySearchHandler(systemName, xmlNamespaceURI, result);
}

static void p1(String m) {
  System.out.println(m);
  }
}

/**
 * An implementation allowing search of CCR by the SystemName.
 */
class CCRSystemNameSearchHandler implements SearchHandler {
  Collection _resultCollection;
  String _systemName;
  public CCRSystemNameSearchHandler(String systemName, Collection resultCollection) {
    _resultCollection = resultCollection;
    _systemName = systemName;
  }
  public boolean carryOn() { return true; }
  public boolean satisfied(Object obj) {
    if (obj==null) return false;
    CCR ccr = (CCR)obj;
    return ccr.getSystemName().equalsIgnoreCase(_systemName);
  }
  public void add(Object obj) {
    _resultCollection.add(obj);
  }


}; // CCRSystemNameSearchHandler
/**
 * An implementation allowing search of CCR by the SystemName and NamespaceURI.
 * There can be at most one matching CCR.
 * This handler is only applicable within a FE.
 */
class CCRKeySearchHandler implements SearchHandler {
  boolean found=false;
  CCR[] _result;
  String _systemName;
  URL _xmlNamespaceURI;
  String _urlString;

  /**
   * @param systemName
   * @param xmlNamespaceURI
   * @param result A single element array to hold the result.
   */
  public CCRKeySearchHandler(String systemName, URL xmlNamespaceURI,
                                    CCR[] result) {
    _systemName = systemName;
    _xmlNamespaceURI = xmlNamespaceURI;
    _urlString = _xmlNamespaceURI.toString();
    _result = result;
    _result[0] = null;
  }
  public boolean carryOn() { return !found; }
  public boolean satisfied(Object obj) {
    if (obj==null) return false;
    CCR ccr = (CCR)obj;
    return ccr.getSystemName().equalsIgnoreCase(_systemName) &&
           ccr.getXMLNameSpaceURI().equalsIgnoreCase(_urlString) ;
  }
  public void add(Object obj) {
    _result[0] = (CCR)obj;
    found = true;
  }

}; // CCRSystemNameSearchHandler
```

7. FCRSchemaImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.*;
import java.lang.reflect.*;
import java.beans.*;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;




/**
 * @author LSC
 * @version 1.0
 */

public class FCRSchemaImpl extends SchemaImpl
                           implements FCRSchema {
```

```
  /**
   * The list of top level attributes of this schema.
   */
  List _attributes = new ArrayList(10);

  /**
   * Public no-arg constructor required by Castor Unmarshaller.
   */
  public FCRSchemaImpl() {
  }
}
```

8. FEImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.Iterator;
import java.util.ArrayList;
import java.util.List;
import java.util.Arrays;

import java.net.URL;



import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

/**
 *
 * @author LSC
 * @version 1.0
 */

public class FEImpl implements FE {

  /**
   * Name of this FE.
   */
  protected String _feName = "";
  /**
   * The FIOM that this FE belongs to.
   */
```

```
private FIOM _fiom = null;

/**
 * The FEV of this FE.
 */
protected FEV _rootFEV = null;
/**
 * The parent of this FE.
 */
protected FE _parent = null;



/**
 * The list of child FEVs of this FEV.
 */
protected List _childList = new ArrayList();



/**
 * Public access required for restoration using Castor Unmarshaller.
 */
public FEImpl() {
}

/**
 * Creates a FEImpl with specified name that belongs to the specified FIOM.
 * Package-level access only.
 */
FEImpl(FIOM fiom, FE parent, String feName) {
  setFIOM(fiom);
  setFEName(feName);
  setParent(parent);
}

/**
 * Returns the FIOM that this FE belongs to.
 * @return Returns the FIOM that this FE belongs to.
 */
public FIOM getFIOM() {
  return _fiom;
}
/**
 * Sets the FIOM that this FE belongs to,
 * should only be called from
 * FIOMImpl.setFE() during restoration from storage,
 * or from this class's constructor.
 * therefore, it is package-level access.
 */
void setFIOM(FIOM v) {
  _fiom = v;
}

/**
 * Gets the name of this FE.
 */
public String getFEName() {
  return _feName;
}
/**
 * Sets the name of this FE.
 * Public access required for restoration from storage.
 */
public void setFEName(String v) {
  _feName = v;
}

public FEV getRootFEV() {
  return _rootFEV;
}

/**
 * Sets the FEV of this FE.
 * Only to be used in the constructor and
 * to restore from persistent storage by Castor Unmarshaller.
 *
 */
public void setRootFEV(FEV v) {
  _rootFEV = v;
  ((FEVImpl)_rootFEV).setFE(this);
}

public FEV findFEV( String fevName ) {
  if ( _rootFEV.getFEVName().equalsIgnoreCase(fevName) )
    return _rootFEV;
  return _rootFEV.findDescendent(fevName, true);
}
```

```
public boolean fevExists(String fevName) {
  return findFEV(fevName)!=null;
}

public FEV createRootFEV(String fevName) {
  _rootFEV = new FEVImpl( null, this, fevName );
  return _rootFEV;
}


/**
 * Equals if and only if names are equal and getFIOM()==o.getFIOM().
 */
public boolean equals(Object o) {
  if (o == null)
    return false;
  if (o instanceof FE) {
    FE fe = (FE)o;
    return getFEName().equalsIgnoreCase(fe.getFEName()) &&
           getFIOM()==fe.getFIOM();
           // cannot use getFIOM().equals() here,
           // else will fall into infinite loop
  }
  else
    return false;
}

public FE getParent() {
  return _parent;
}

/**
 * Sets the parent of this FE, the caller is responsible for updating the
 * child list of the parent.
 * Private access, to be called only from
 *  FEImpl.setChild(), FEImpl.createChild() and FEImpl.removeChild().
 */
private void setParent(FE parent) {
  _parent = parent;
}

public int childCount() {
  return _childList.size();
}

public FE[] getChild() {
  return (FE[])_childList.toArray( new FE[ childCount() ] );
}
/**
 * Sets the list of children with the specified array.
 * Generally, to be called only from the Castor Unmarshaller.
 */
public void setChild( FE[] v ) {
  _childList.clear();
  _childList.addAll( Arrays.asList(v) );
  FEImpl feImpl;
  for (int i=0; i<v.length; i++) {
    feImpl = (FEImpl)(v[i]);
    feImpl.setParent(this);
    feImpl.setFIOM(this.getFIOM());
  }
}

public FE newChild(String childFEName) throws DuplicateKeyException {
  if (hasDescendent(childFEName, true)) {
    throw new DuplicateKeyException(childFEName + " already exists.");
  }
  FEImpl fe = new FEImpl(this.getFIOM(), this, childFEName);
  _childList.add(fe);
  return fe;
}

public boolean hasDescendent(String descendentName, boolean recurseTree) {
  return null != findDescendent(descendentName, recurseTree);
}

public FE findDescendent(String descendentName, boolean recurseTree) {
  FE[] children = getChild();
  FE result;
  FE curr;
  for (int i=0; i<children.length; i++) {
    curr = children[i];
    if (curr.getFEName().equalsIgnoreCase(descendentName))
      return curr;
    if (recurseTree) {
      result = curr.findDescendent(descendentName, true);
      if (result!=null)
        return result;
    }
  }
  return null;
}
public void findCCR(SearchHandler handler) {
  if (getRootFEV()!=null)
    getRootFEV().findCCR(handler);
  if (handler.carryOn())
    findDescendentCCR(handler);
}

public void findDescendentCCR(SearchHandler handler) {
  FE[] children = getChild();
  FE curr;
  for (int i=0; i<children.length; i++) {
    curr = children[i];
    curr.findCCR(handler);
    /*
    if (handler.satisfied(curr.getRootFEV().getFCR()))
      handler.add(curr.getRootFEV().getFCR());
    if (handler.carryOn())
      curr.findDescendentCCR(handler);
    */
  }
}
public CCR[] findCCR(String systemName) {
  java.util.Set resultSet = new java.util.HashSet();
  findCCR( FCRImpl.makeCCRSystemNameSearchHandler( systemName, resultSet ) );
  return (CCR[]) resultSet.toArray(new CCR[resultSet.size()]);
}

public CCR findCCR(String systemName, URL xmlNameSpaceURI) {
  CCR[] result = new CCR[1];
  findCCR( FCRImpl.makeCCRKeySearchHandler(systemName, xmlNameSpaceURI, result) );
  return result[0];
  /*
  CCR result = null;
  if (getRootFEV()!=null)
    result = getRootFEV().findCCR(systemName, xmlNameSpaceURI);
  if (result==null)
```

```
      return findDescendentCCR(systemName, xmlNameSpaceURI);
    else
      return result;
    */
  }

  public CCR findDescendentCCR(String systemName, URL xmlNameSpaceURI) {
    FE[] children = getChild();
    CCR result;
    FE curr;
    for (int i=0; i<children.length; i++) {
      curr = children[i];
      result = curr.findCCR(systemName, xmlNameSpaceURI);
      if (result!=null)
        return result;
    }
    return null;
  }

}
```

9. FEVImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.List;
import java.util.Arrays;
import java.util.Iterator;
import java.util.ArrayList;
import java.net.URL;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;


/**
 * A reference implementation of a Federation Entity View.
 * @author LSC
 * @version 1.0
 */

public class FEVImpl implements FEV {
```

```
/**
 * The parent of this FEV.
 */
protected FEV _parent = null;

/**
 * The list child FEVs of this object.
 */
protected List _childList = new ArrayList();

/**
 * The name of this FEV.
 */
protected String _fevName = "";

/**
 * The FE of this FEV.
 */
protected FE _fe = null;

/**
 * The FCR corresponding to this FEV.
 */
protected FCR _fcr = null;

/**
 * Public constructor required for Castor Marshal and Unmarshal.
 */
public FEVImpl() {
}

/**
 * Creates an FEVImpl instance with the specified name and FE.
 */
FEVImpl( FEV parent, FE fe, String fevName ) {
  setFE(fe);
  setFEVName(fevName);
  setParent(parent);
}

public FCR createFCR(String fcrName) {
  FCR result = new FCRImpl(this, fcrName);
  setFCR(result);
  return result;
}


public String getFEVName() {
  return _fevName;
}
/**
 * Sets the name of this FEV.
 * Only to be used to restore this object from persistent storage.
 */
public void setFEVName(String v) {
  _fevName = v;
}

public FE getFE() {
  return _fe;
}

/**
 * Sets the FE of this FEV.
 * Only to be used in the constructor and
 * to restore this object from persistent storage.
 */
public void setFE(FE fe) {
  _fe = fe;
}

public FCR getFCR() {
  return _fcr;
}

/**
 * Sets the FCR of this FEV.
 * To be used in the constructor,
 * Castor Unmarshaller and
 * by FEImpl.setFCR() to restore this object from persistent storage.
 */
public void setFCR(FCR fcr) {
  _fcr = fcr;
  ((FCRImpl)_fcr).setFEV(this);
}
```

```
public FEV getParent() {
  return _parent;
}

/**
 * Sets the parent of this FEV, the caller is responsible for updating the
 * child list of the parent.
 * Private access, to be called only from
 *  FEVImpl.setChild(), FEVImpl.createChild() and FEVImpl.removeChild().
 */
private void setParent(FEV parent) {
  _parent = parent;
}

public int childCount() {
  return _childList.size();
}

public FEV[] getChild() {
  return (FEV[])_childList.toArray( new FEV[ childCount() ] );
}

/**
 * Sets the list of children with the specified array.
 * Sets the Parent and FE properties of each child appropriately.
 * Generally, to be called only from the Castor Unmarshaller.
 */
public void setChild( FEV[] v ) {
  _childList.clear();
  _childList.addAll( Arrays.asList(v) );
  FEVImpl fevImpl;
  for (int i=0; i<v.length; i++) {
    fevImpl = (FEVImpl)(v[i]);
    fevImpl.setParent(this);
    fevImpl.setFE(this.getFE());
  }
}

/**
 * Create a new child FEV for this FEV and returns the newly created child.
 * @return Returns the newly created child FEV.
 */
```

```
public FEV newChild(String childFEVName) throws DuplicateKeyException {
  if (hasDescendent(childFEVName, true)) {
    throw new DuplicateKeyException(childFEVName + " already exists.");
  }
  FEVImpl fev = new FEVImpl(this, this.getFE(), childFEVName);
  _childList.add(fev);
  return fev;
}

public boolean hasDescendent(String descendentName, boolean recurseTree) {
  return null != findDescendent(descendentName, recurseTree);
}

public FEV findDescendent(String descendentName, boolean recurseTree) {
  FEV[] children = getChild();
  FEV result;
  FEV curr;
  for (int i=0; i<children.length; i++) {
    curr = children[i];
    if (curr.getFEVName().equalsIgnoreCase(descendentName))
      return curr;
    if (recurseTree) {
      result = curr.findDescendent(descendentName, true);
      if (result!=null)
        return result;
    }
  }
  return null;
}

public void findDescendentCCR( SearchHandler handler ) {
  // depth-first search
  FEV[] children = getChild();
  FEV curr;
  for (int i=0; i<children.length && handler.carryOn(); i++) {
    curr = children[i];
    curr.findCCR(handler);
  }
}
public void findCCR(SearchHandler handler) {
  if (getFCR()!=null)
    getFCR().findCCR(handler);
  if (handler.carryOn())
```

```
    findDescendentCCR(handler);
}

public CCR[] findCCR( String systemName ) {
  java.util.HashSet resultSet = new java.util.HashSet();
  findCCR( FCRImpl.makeCCRSystemNameSearchHandler(systemName, resultSet) );
  return (CCR[]) resultSet.toArray( new CCR[resultSet.size()] );
}

public CCR findCCR(String systemName, URL xmlNameSpaceURI) {
  CCR[] result = new CCR[1];
  findCCR( FCRImpl.makeCCRKeySearchHandler(systemName, xmlNameSpaceURI, result) );
  return result[0];
  /*
  CCR result = null;
  if (getFCR()!=null)
    result = getFCR().findCCR(systemName, xmlNameSpaceURI);
  if (result==null)
    return findDescendentCCR(systemName, xmlNameSpaceURI);
  else
    return result;
  */
}

public CCR findDescendentCCR(String systemName, URL xmlNameSpaceURI) {
  FEV[] children = getChild();
  CCR result;
  FEV curr;
  for (int i=0; i<children.length; i++) {
    curr = children[i];
    result = curr.findCCR(systemName, xmlNameSpaceURI);
    if (result!=null)
      return result;
  }
  return null;
}

/**
 * Equals if and only if getFEVName().equalsIgnoreCase(o.getFEVName())
 * and getFE()==o.getFE.
 */
public boolean equals(Object o) {
  if (o==null)
```

```
      return false;
    if ( o instanceof FEV ) {
      FEV fev = (FEV) o;
      return getFEVName().equalsIgnoreCase(fev.getFEVName()) &&
             getFE()==fev.getFE() ;
    }
    else
      return false;
  }

  public int hashCode() {
    StringBuffer sb = new StringBuffer();
    if (getFE()!=null)
      sb.append( getFE().getFEName() );
    sb.append("/" + getFEVName());
    return sb.toString().toLowerCase().hashCode();
  }
}
```

10. FIOMImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;
import java.util.Map;
import java.util.StringTokenizer;
import java.net.URL;

import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;
import mil.navy.nps.cs.oomi.*;

/**
 *
 * @author LSC
 * @version 1.0
 */

public class FIOMImpl implements FIOM {

  /**
   * The name of this fiom.
   */
  protected String _fiomName = "";

  /**
   * The FEs in this FIOM.
   */
  protected List _feList = new ArrayList();

  /**
   * Holds the list of translations known to this FIOM.
   * Each entry has a key made of the fully qualified name of the
   * FCRInstance's class and the CCRInstance's class, separated by a comma.
   * Each entry's value is the fully qualified class name of the specific
   * descendent of AbstractTranslation.
   */
  protected Map _translationMap = new HashMap();

  /**
   * The factory for the implementation of this FIOM.
   * This field is initialised whenever a new instance is constructed.
   */
  protected FIOMFactory _factory = null;

  /**
   * The list of unregistered CCRs corresponding to this FIOM.
   */
  protected Set _unregisteredCcrSet = new HashSet();

  /**
   * Initialisation, to be called from all constructors.
   */
  private void init() {
    _factory = FactoryImpl.factory();
  }

  /**
   * Required for save/load to/from file.
   */
  public FIOMImpl() {
    init();
  }

  /**
   * No arguments constructor needed for Marshalling by Castor.
   */
  public FIOMImpl(String name) {
    init();
    _fiomName = name;
  }

  public String getFIOMName() {
    return _fiomName;
  }

  /**
   * Sets the name of this FIOM.
   * Only to be used to restore this object from persistent storage.
   */
  public void setFIOMName(String name) {
    _fiomName = name;
  }

  public int feCount() {
    return _feList.size();
  }

  public FE[] getFE() {
    return (FE[])_feList.toArray(new FE[feCount()]);
  }

  /**
   * Sets the array of FEs in this FIOM, used to restore this object from
   * persistent storage.  Calls setFIOM() on each element in v.
   *
   * @param v The array of FEs to be set for this FIOM, each element's
   *          fiom property is set to this object.
   */
  public void setFE(FE[] v) {
    _feList.clear();
    _feList.addAll(Arrays.asList(v));
    for (int i=0; i<v.length; i++) {
      ((FEImpl) v[i]).setFIOM(this);
    }
  }
  public FE findFE(String feName) {
    Iterator iter = _feList.iterator();
    FE fe;
    while (iter.hasNext()) {
      fe = (FE)iter.next();
      if ( fe.getFEName().equalsIgnoreCase(feName) )
        return fe;
    }
    return null;
  }

  public boolean feExists(String feName) {
    return findFE(feName) != null;
  }

  public FE newFE(String feName) throws DuplicateKeyException {
    if ( feExists(feName) )
      throw new DuplicateKeyException(feName + " already exists.");
    FEImpl fe = new FEImpl( this, null, feName );
    _feList.add(fe);
    return fe;
  }

  public CCR findCCR(String systemName, URL xmlNameSpaceURI) {
    FE[] feArray = getFE();
    CCR result;
    for (int i=0; i<feArray.length; i++) {
      result = feArray[i].findCCR(systemName, xmlNameSpaceURI);
      if (result!=null)
        return result;
    }
    return null;
  }


  public CCR newCCR(String systemName, String ccrName, URL xmlNameSpaceURI)
                                                throws DuplicateKeyException {
    if ( ccrExists(systemName, ccrName) )
      throw new DuplicateKeyException(ccrName + " already exists for system: " + systemName);
    if ( ccrExists(systemName, xmlNameSpaceURI) )
      throw  new  DuplicateKeyException(xmlNameSpaceURI  +  "  already  exists  for  system:  "  +
systemName);
    //CCRImpl ccr = new CCRImpl(null, systemName, ccrName);
    CCR ccr = factory().makeCCR(null, systemName, ccrName);
    if ( xmlNameSpaceURI!=null)
      ccr.setXMLNameSpaceURI( xmlNameSpaceURI.toString() );
    addUnregisteredCCR(ccr);
    return ccr;
  }// CCR newCCR(String, String, URL)



  public void registerCCR(FCR fcr, CCR ccr) {
    if (_unregisteredCcrSet.contains(ccr)) {
      ccr.joinFCR(fcr);
      _unregisteredCcrSet.remove(ccr);
    }
  }

  public void addUnregisteredCCR( CCR newCCR ) {
      _unregisteredCcrSet.add(newCCR);
  }
  public CCR[] getUnregisteredCCR( ) {
    return (CCR[]) _unregisteredCcrSet.toArray(
                        new CCR[unregisteredCCRCount()] );
  }
  public int unregisteredCCRCount() {
    return _unregisteredCcrSet.size();
  }
  public CCR findUnregisteredCCR(String systemName, String ccrName) {
    Iterator iter = _unregisteredCcrSet.iterator();
    CCR ccr;
    while (iter.hasNext()) {
      ccr = (CCR)iter.next();
      if ( ccr.getSystemName().equalsIgnoreCase(systemName) &&
           ccr.getCCRName().equalsIgnoreCase(ccrName) )
        return ccr;
    }
    return null;
  }
  public CCR findUnregisteredCCR(String systemName, URL xmlNameSpaceURI) {
    if (xmlNameSpaceURI==null)
      return null;
    Iterator iter = _unregisteredCcrSet.iterator();
    CCR ccr;
    String urlString = xmlNameSpaceURI.toString();
    while (iter.hasNext()) {
      ccr = (CCR)iter.next();
      if ( ccr.getSystemName().equalsIgnoreCase(systemName) &&
           ccr.getXMLNameSpaceURI().equals(urlString) )
        return ccr;
    }
    return null;
  }

  public boolean ccrExists(String systemName, String ccrName) {
    return findUnregisteredCCR(systemName, ccrName)!=null;
  }
  public boolean ccrExists(String systemName, URL xmlNameSpaceURI) {
    return findUnregisteredCCR(systemName, xmlNameSpaceURI)!=null;
  }

  public boolean equals(Object o) {
    if (o==null)
      return false;
    if (o instanceof FIOM) {
      FIOM fiom = (FIOM) o;
      if (!getFIOMName().equalsIgnoreCase(fiom.getFIOMName()))
        return false;
      else
        return Arrays.equals(getFE(), fiom.getFE()) ;
    }
    else
      return  false;
  }

  public void addTranslation( String fcrClassName, String ccrClassName,
                              String translationClassName ) {
    _translationMap.put( fcrClassName + "," + ccrClassName,
                         translationClassName );
  }
  public String findTranslation( String fcrClassName, String ccrClassName ) {
    String result = (String)_translationMap.get( fcrClassName + "," + ccrClassName );
    if (result==null)
      return null;
    if (result.length()==0)
```

```
    return null;
  else
    return result;
}
/**
 * For use by Castor Marshaller.
 */
public String[] getTranslations() {
  String[] result = new String[_translationMap.size()];
  Set entries = _translationMap.entrySet();
  Iterator iter = entries.iterator();
  Map.Entry entry;
  int i = 0;
  while (iter.hasNext()) {
    entry = (Map.Entry) iter.next();
    result[i] = (String)entry.getKey() + "=" + (String)entry.getValue();
    i++;
  }
  return result;
}
/**
 * For use by Castor Unmarshaller.
 */
public void setTranslations(String[] v) {
  _translationMap.clear();
  StringTokenizer tokenizer;
  String key;
  String val;
  for (int i=0; i<v.length; i++) {
    tokenizer = new StringTokenizer( v[i], "=" );
    if (tokenizer.hasMoreTokens())
      key = tokenizer.nextToken();
    else
      continue;
    if (tokenizer.hasMoreTokens())
      val = tokenizer.nextToken();
    else
      continue;
    if (key.length()==0 || val.length()==0)
      continue;
    _translationMap.put( key, val );
  }
}
```

```
  public FIOMFactory factory() {
    return _factory;
  }


  public TypeName makeTypeName( String typeName ) {
    return _factory.makeTypeName(typeName);
  }
  public TypeName makeTypeName( Class theClass ) {
    return _factory.makeTypeName(theClass);
  }
  public TypeName makeArrayTypeName( String typeName, int dimensionCount ){
    return _factory.makeArrayTypeName(typeName, dimensionCount);
  }
  public TypeName makeArrayTypeName( Class theClass, int dimensionCount ){
    return _factory.makeArrayTypeName(theClass, dimensionCount);
  }

  static void p(String m) {
    System.out.println(m);
  }

}
```

11. OOMIDatabaseImpl.java

```
package mil.navy.nps.cs.oomi.impl;


import java.util.*;
import java.io.FileWriter;
import java.io.FileReader;
import java.io.IOException;
import java.io.File;

import org.exolab.castor.xml.Marshaller;
import org.exolab.castor.xml.Unmarshaller;
import org.exolab.castor.xml.MarshalException;
import org.exolab.castor.xml.ValidationException;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.exceptions.*;
import mil.navy.nps.cs.oomi.fiom.*;



/**
 * A default implementation of OOMIDatabase.
 * Relies on the Castor Marshaller and Unmarshaller to handle
 * loading/saving from/to persistent storage (an XML file).
 *
 * @author LSC
 * @version 1.0
 */

public class OOMIDatabaseImpl implements OOMIDatabase {

  /**
   * The list of FIOM's in this database.
   */
  List _fiomList = new ArrayList();

  /**
   * This class should only be instantiated using
   * <a href="#saveToFile(String)"> saveToFile(String xmlFilename) </a>.
   */
  public OOMIDatabaseImpl() {
  }

  /**
   * Saves this OOMIDatabase to an XML file.
   */
  public void saveToFile(String xmlFilename) throws IOException,
                                                    ValidationException,
                                                    MarshalException {
    FileWriter writer = new FileWriter(xmlFilename);
    Marshaller.marshal(this, writer);
  }

  /**
   * Creates a new OOMIDatabase object from data in the specified XML file.
   * If the XML file does not exist, the returned object is
   *         set to default values that represents an empty database.
   *
   * @param xmlFilename The XML file where the data is stored.
   * @return Returns a new OOMIDatabase with content restored from the specified
   *         XML file. If the XML file does not exist, the returned object is
   *         set to default values that represents an empty database.
   */
  public static OOMIDatabase loadFromFile(String xmlFilename)
                                            throws IOException,
                                                    ValidationException,
                                                    MarshalException {

    if (new File(xmlFilename).exists()) {
      FileReader reader = new FileReader(xmlFilename);
      return (OOMIDatabase) Unmarshaller.unmarshal(OOMIDatabaseImpl.class, reader);
    }
    else
      return new OOMIDatabaseImpl();
  }

  public int fiomCount() {
    return _fiomList.size();
  }

  public FIOM[] getFIOM() {
    return (FIOM[]) _fiomList.toArray(new FIOM[fiomCount()]);
  }

  /**
   * Sets the list of FIOM's from the specified value.
   * For use by the persistence storage to restore this object.
   */
  public void setFIOM( FIOM[] value ) {
    _fiomList.clear();
    _fiomList.addAll(Arrays.asList(value));
  }
  public static void p(String m) {
    System.out.println(m);
  }
  public FIOM newFIOM( String fiomName ) throws DuplicateKeyException {
    FIOM result = null;
    if (fiomExists(fiomName)) {
      throw new DuplicateKeyException(fiomName + " already exists.");
    }
    result = new FIOMImpl(fiomName);
    _fiomList.add(result);
    return result;
  }
  public boolean fiomExists( String fiomName ) {
    return findFIOM(fiomName) != null;
  }

  public FIOM findFIOM( String fiomName ) {
    FIOM result = null;
    Iterator iter = _fiomList.iterator();
    FIOM curr;
    while (iter.hasNext()) {
      curr = (FIOM) iter.next();
      if (fiomName.equalsIgnoreCase(curr.getFIOMName())) {
        result = curr;
        break;
      }
    }
    return result;
  }

  public boolean equals(Object o) {
    if (o == null)
      return false;
    if ( o instanceof OOMIDatabaseImpl ) {
      OOMIDatabaseImpl db = (OOMIDatabaseImpl) o;
      return Arrays.equals( getFIOM(), db.getFIOM() );
    }
    else
      return false;
  }

}
```

12. SchemaImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.List;
import java.util.Arrays;
import java.util.Iterator;
import java.util.ArrayList;
import java.util.WeakHashMap;
import java.util.Map;
import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

/**
 * @author LSC
 * @version 1.0
 */

public class SchemaImpl implements Schema {



  /**
   * The list of top level attributes of this schema.
   */
  protected List _attributes = new ArrayList();

  /**
   * A cache to attempt to speed up findAttribute().
   */
  private Map _attributeMap = new WeakHashMap();

  /**
   * The fully qualified name of the class that this schema describes.
   */
  protected TypeName _type = null;

  /**
   * Public no-arg constructor needed by Castor Unmarshaller.
   */
  public SchemaImpl() {
  }

  public TypeName getType() { return _type; }
  public void   setType(TypeName type) { this._type = type; }

  public int attributeCount() { return _attributes.size(); }

  public Attribute[] getAttribute() {
    return (Attribute[]) _attributes.toArray( new Attribute[attributeCount()] );
  }
  public void setAttribute( Attribute[] value ) {
```

```
  _attributes.clear();
  _attributes.addAll( Arrays.asList(value) );
}
public void addAttribute( Attribute attr ) {
  _attributes.add(attr);
}
public Attribute addAttribute( String name, TypeName type ) {
  Attribute result = FactoryImpl.factory().makeAttribute( null, name, type );
  _attributes.add(result);
  return result;
}
public void removeAttribute( Attribute attr ) {
  // remove all entries in the cache
  _attributeMap.clear();
  if ( _attributes.contains(attr) )
    _attributes.remove(attr);
  else {
    boolean done = false;
    Iterator i = _attributes.iterator();
    Attribute curr;
    while (i.hasNext() && !done) {
      curr = (Attribute)i.next();
      done = curr.removeDescendent(attr);
    }
  }
}
public Attribute findAttribute( Attribute attr ) {
  Iterator i = _attributes.iterator();
  Attribute curr;
  Attribute c;
  while (i.hasNext()) {
    curr = (Attribute) i.next();
    if (curr.equals(attr))
      return curr;
    c = curr.findDescendent(attr);
    if (c!=null) {
      return c;
    }
  }
  return null;
}
public void enumerateAttributes( List result ) {
  Iterator i = _attributes.iterator();
```

```
    Attribute attr;
    while (i.hasNext()) {
      attr = (Attribute) i.next();
      result.add(attr);
      attr.enumerateDescendents( result );
    }
  }
  public Attribute[] enumerateAttributes(  ) {
    List list = new ArrayList(20);
    enumerateAttributes(list);
    return (Attribute[]) list.toArray( new Attribute[list.size()] );

  }

}
```

13. TypeNameImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.Set;
import java.util.HashSet;
import java.util.Arrays;

import mil.navy.nps.cs.oomi.fiom.TypeName;

/**
 * @author LSC
 * @version 1.0
 */

public class TypeNameImpl implements TypeName {

  /**
   *
   * @invariant _typeName != null
   */
  String _typeName = "";

  /**
   * Public no-argument constructor required by Castor.
   */
  public TypeNameImpl() {
  }

  public TypeNameImpl(String typeName) {
    if (typeName==null)
      setTypeName("");
    else
      setTypeName(typeName);
  }

  public String getTypeName() { return _typeName; }
  public void   setTypeName(String value) {
    if (value==null)
      _typeName = "";
    else
      _typeName = value;
  }

  public boolean equals(Object obj) {
    return toString().equals( ((TypeName)obj).toString() );
  }
  public int hashCode() {
    return toString().hashCode();
  }
  protected final char ARRAY_INDICATOR = '[';
  protected final char CLASS_INDICATOR = 'L';
  protected final Set _primitiveTypes = new HashSet(8);
    {
      _primitiveTypes.addAll( Arrays.asList(PRIMITIVE_NAMES) );
    }
  public boolean isArray() {
    if (_typeName.length()==0)
      return false;
    else
      return _typeName.charAt(0)==ARRAY_INDICATOR;
  }
  public boolean isPrimitive() {
    if (_typeName.length()==0)
      return false;
    else {
      if (isArray())
        return false;
      else {
        return _primitiveTypes.contains(_typeName);
      }
    }
  }
  public String toString() {
    return _typeName;
  }
  public TypeName copy() {
    return FactoryImpl.factory().makeTypeName( toString() );
  }
  protected Object clone() {
    return copy();
  }

}
```

14. ZzzTest_AttributeImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import junit.framework.TestCase;
import java.util.List;
import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.ArrayList;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

public class ZzzTest_AttributeImpl extends TestCase
{
  FIOMFactory _factory = FactoryImpl.newFIOMFactory();

    public ZzzTest_AttributeImpl(String Name_)
    {
        super(Name_);
    } //public ZzzTest_AttributeImpl(String Name_)

    protected void setUp()
    {
    } //protected void setUp()
    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_AttributeImpl.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)

  public static void p(String m) {
    System.out.println(m);
  }

  public void test_constructor_getset_Parent_Name_Type_1() {
    Attribute parent = new AttributeImpl();
    Attribute attr = _factory.makeAttribute(parent, "The name",
                                            _factory.makeTypeName("The Type") );
    assertTrue( parent.getParent() == null
                && attr.getParent() == parent
                && attr.getName().equals("The name")
                && attr.getType().equals(_factory.makeTypeName("The Type"))
                );
  }

  public void test_childCount_1() throws Exception {
    Attribute  attr  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type") );
    attr.newChild(      "name1",      _factory.makeTypeName("type")      );//.newChild("name1-
1",_factory.makeTypeName("type1-1"));
    attr.newChild( "name2", _factory.makeTypeName("type"));
    assertEquals( 2, attr.childCount() );
  }

  public void test_getChild_1() throws Exception {
    Attribute  attr  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type"));
    Attribute[] attrs = new Attribute[] { attr.newChild("1",_factory.makeTypeName("A")),
                                          attr.newChild("2",_factory.makeTypeName("B")) };
    assertTrue( Arrays.equals(attrs, attr.getChild()) );
  }
  /**
   * Calls setChild() once.
   */
  public void test_setChild_1() throws Exception {
    AttributeImpl   attr   =   (AttributeImpl)   _factory.makeAttribute(   null,   "The   name",
_factory.makeTypeName("The Type") );
    Attribute[]    attrs    =    new    Attribute[]    {    _factory.makeAttribute(    null,
"1",_factory.makeTypeName("A")),
                                        _factory.makeAttribute(                         null,
"2",_factory.makeTypeName("B") ) };
    attr.setChild(attrs);
    assertTrue( Arrays.equals(attrs, attr.getChild()) );
  }
  /**
   * Calls setChild() twice to ensure replacement by the second call.
   */
  public void test_setChild_2() throws Exception {
    AttributeImpl   attr   =   (AttributeImpl)   _factory.makeAttribute(   null,   "The   name",
_factory.makeTypeName("The Type") );
    Attribute[]    attrs    =    new    Attribute[]    {    _factory.makeAttribute(    null,
"1",_factory.makeTypeName("A") ),
                                        _factory.makeAttribute(                         null,
"2",_factory.makeTypeName("B") ) };
    attr.setChild(attrs);
    attr.setChild(attrs);
    assertTrue( Arrays.equals(attrs, attr.getChild()) );
  }

  /**
   * Find child, should not match with itself.
   */
  public void test_findChild_1() throws Exception {
    Attribute  attr  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type") );
    Attribute attr_1 = attr.newChild( "name1", _factory.makeTypeName("type"));
    Attribute attr_1_1 = attr_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute attr_2 = attr.newChild( "name2", _factory.makeTypeName("type"));
    Attribute attr_3 = attr.newChild( "The name", _factory.makeTypeName("type"));
    assertTrue( attr.findChild( "The name" ) == attr_3 );
  }
  /**
   * Find child, should not find grandchildren.
   */
  public void test_findChild_2() throws Exception {

    Attribute  attr  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type"));
    Attribute attr_1 = attr.newChild( "name1", _factory.makeTypeName("type"));
    Attribute attr_1_1 = attr_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute attr_2 = attr.newChild( "name2", _factory.makeTypeName("type"));
    Attribute attr_3 = attr.newChild( "The name", _factory.makeTypeName("type"));
    assertTrue( attr.findChild( "name1-1" ) == null );
  }
  /**
   * Find descendent.
   */
  public void test_findDescendent_Attribute_1() throws Exception {

    Attribute  attr  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type"));
    Attribute attr_1 = attr.newChild( "name1", _factory.makeTypeName("type"));
    Attribute attr_1_1 = attr_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute attr_2 = attr.newChild( "name2", _factory.makeTypeName("type"));
    Attribute attr_3 = attr.newChild( "The name", _factory.makeTypeName("type"));
    Attribute attr_4 = attr.newChild( "The name 4", _factory.makeTypeName("type"));

    Attribute  a  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type"));
    Attribute a_1 = a.newChild( "name1", _factory.makeTypeName("type"));
    Attribute a_1_1 = a_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute a_2 = a.newChild( "name2", _factory.makeTypeName("type"));
    Attribute a_3 = a.newChild( "The name", _factory.makeTypeName("type"));
    Attribute a_4 = a.newChild( "The name 4", _factory.makeTypeName("type"));
    assertTrue( attr_4 == attr.findDescendent( a_4 ) );
  }
  /**
   * Find descendent.
   */
  public void test_findDescendent_Attribute_2() throws Exception {
    Attribute  attr  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type"));
    Attribute attr_1 = attr.newChild( "name1", _factory.makeTypeName("type"));
    Attribute attr_1_1 = attr_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute attr_2 = attr.newChild( "name2", _factory.makeTypeName("type"));
    Attribute attr_3 = attr.newChild( "The name", _factory.makeTypeName("type"));
    Attribute attr_4 = attr.newChild( "The name 4", _factory.makeTypeName("type"));

    Attribute  a  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type"));
    Attribute a_1 = a.newChild( "name1", _factory.makeTypeName("type"));
    Attribute a_1_1 = a_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute a_2 = a.newChild( "name2", _factory.makeTypeName("type"));
    Attribute a_3 = a.newChild( "The name", _factory.makeTypeName("type"));
    Attribute a_4 = a.newChild( "The name 4", _factory.makeTypeName("type"));
    //assertTrue( attr_1_1 == attr.findDescendent( a_1 ) );
    assertTrue( attr_1_1 == attr.findDescendent( a_1_1 ) );
  }
  /**
   * Find descendent.
   */
  public void test_findDescendent_Attribute_3() throws Exception {
    Attribute  attr  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type"));
    Attribute attr_1 = attr.newChild( "name1", _factory.makeTypeName("type"));
    Attribute attr_1_1 = attr_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute attr_2 = attr.newChild( "name2", _factory.makeTypeName("type"));
    Attribute attr_3 = attr.newChild( "The name", _factory.makeTypeName("type"));
    Attribute attr_4 = attr.newChild( "The name 4", _factory.makeTypeName("type"));


    Attribute  a  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type"));
    Attribute a_1 = a.newChild( "name1", _factory.makeTypeName("type"));
    Attribute a_1_1 = a_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute a_2 = a.newChild( "name2", _factory.makeTypeName("type"));
    Attribute a_3 = a.newChild( "The name", _factory.makeTypeName("type"));
    Attribute a_4 = a.newChild( "The name 4 not found", _factory.makeTypeName("type"));
    assertTrue( attr.findDescendent( a_4 ) == null );
  }
  /**
   * Tests that copy() properly copies the ancestors.
   */
  public void test_copy_1() throws Exception {
    Attribute  attr  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type"));
    Attribute attr_1 = attr.newChild( "name1", _factory.makeTypeName("type"));
    Attribute attr_1_1 = attr_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute attr_2 = attr.newChild( "name2", _factory.makeTypeName("type"));
    Attribute attr_3 = attr.newChild( "The name", _factory.makeTypeName("type"));
    Attribute attr_4 = attr.newChild( "The name 4", _factory.makeTypeName("type"));
    assertTrue( attr.copy().equals( attr ) && attr_4.copy().equals(attr_4) );
  }
  /**
   * Tests that copy() properly copies the descendents.
   */
  public void test_copy_2() throws Exception {
    Attribute  attr  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type"));
      attr.setMinOccurs(100);
      attr.setMaxOccurs(200);
    Attribute attr_1 = attr.newChild( "name1", _factory.makeTypeName("type"));
    Attribute attr_1_1 = attr_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
      attr_1_1.setMinOccurs(10);
      attr_1_1.setMaxOccurs(20);
    Attribute attr_2 = attr.newChild( "name2", _factory.makeTypeName("type"));
    Attribute attr_3 = attr.newChild( "The name", _factory.makeTypeName("type"));
    Attribute attr_4 = attr.newChild( "The-name-4", _factory.makeTypeName("type"));
    Attribute clone = attr.copy();
    Attribute clone_1_1 = clone.findDescendent(attr_1_1);
    Attribute clone_3 = clone.findDescendent(attr_3);
    assertTrue( clone_1_1!=attr_1_1
                && clone.childCount()==4
                && clone_1_1.equals(attr_1_1)
                && clone_1_1.getParent().equals(attr_1)
                && attr_4.childCount()==0
                && clone.getMinOccurs() == attr.getMinOccurs()
                && clone.getMaxOccurs() == attr.getMaxOccurs()
                && clone_1_1.getMinOccurs() == attr_1_1.getMinOccurs()
                && clone_1_1.getMaxOccurs() == attr_1_1.getMaxOccurs()
                && clone_3.getMinOccurs() == attr_3.getMinOccurs()
                && clone_3.getMaxOccurs() == attr_3.getMaxOccurs()
                );
  }
  /**
   * Duplicate attribute name.
   */
  public void test_newChild_1() throws Exception {
    try {
      Attribute attr = _factory.makeAttribute( null, "The name", _factory.makeTypeName("The
Type"));
      attr.newChild(          "name1",          _factory.makeTypeName("type")).newChild("name1-
1",_factory.makeTypeName("type1-1"));
      attr.newChild( "name2", _factory.makeTypeName("type"));
      attr.newChild( "name2", _factory.makeTypeName("type"));
      fail( UniqueNameViolationException.class.getName() +
            " expected but not thrown." );
    }
    catch (Exception exc) {
      assertEquals( UniqueNameViolationException.class, exc.getClass() );
    }
  }
  /**
   * Duplicate attribute name in grandchild,
   * no exceptions should be encountered.
   */
  public void test_newChild_2() throws Exception {
    try {
      Attribute attr = _factory.makeAttribute( null, "The name", _factory.makeTypeName("The
Type"));
      attr.newChild(          "name1",          _factory.makeTypeName("type")).newChild("name1-
1",_factory.makeTypeName("type1-1"));
      attr.newChild(      "name2",      _factory.makeTypeName("type")).newChild(      "name2",
_factory.makeTypeName("type"));
      assertTrue( true );
    }
    catch (UniqueNameViolationException exc) {
      fail( UniqueNameViolationException.class.getName() +
            " thrown." );
    }
  }
  /**
   * newChild(), tests Parent set.
   */
  public void test_newChild_3() throws Exception {
    Attribute parent = _factory.makeAttribute( null, "The name", _factory.makeTypeName("The
Type"));
    Attribute child = parent.newChild( "name1", _factory.makeTypeName("type"));
    assertTrue( parent == child.getParent() );
  }

  /**
   * Test the set/get of minOccurs, maxOccurs.
   */
  public void test_getset_minOccurs_maxOccurs() {
    Attribute attr = new AttributeImpl();
    attr.setMinOccurs(1);
    attr.setMaxOccurs(10);
    assertTrue( attr.getMinOccurs()==1 &&
                attr.getMaxOccurs()==10 );
  }
  /**
   * Tests isMandatory().
   * getMinOccurs() == 1
   */
  public void test_isMandatory_1_true() {
    Attribute attr = new AttributeImpl();
    attr.setMinOccurs(1);
    attr.setMaxOccurs(10);
    assertTrue( attr.isMandatory() );
  }
  /**
   * Tests isMandatory().
   * getMinOccurs() > 1
   */
  public void test_isMandatory_2_true() {
    Attribute attr = new AttributeImpl();
    attr.setMinOccurs(2);
    attr.setMaxOccurs(10);
    assertTrue( attr.isMandatory() );
  }
  /**
   * Tests isMandatory().
   */
  public void test_isMandatory_1_false() {
    Attribute attr = new AttributeImpl();
    attr.setMinOccurs(0);
    attr.setMaxOccurs(10);
    assertTrue( !attr.isMandatory() );
  }
  /**
   * Tests isArray().
   */
  public void test_isArray_1_true() {
    Attribute attr = _factory.makeAttribute( null, "abc", _factory.makeArrayTypeName("def",1)
);
    attr.setMinOccurs(1);
    attr.setMaxOccurs(10);
    assertTrue( attr.isArray() );
  }
  /**
   * Tests isArray(). getMaxOccurs()>1.
   */
  public void test_isArray_1_false() {
    Attribute attr = _factory.makeAttribute( null, "abc", _factory.makeTypeName("def") );
    attr.setMinOccurs(1);
    attr.setMaxOccurs(2);
    assertTrue( !attr.isArray() );
  }

  public void test_equals_1() {
    Attribute attr1 = new AttributeImpl(null, "name", _factory.makeTypeName("type"));
    Attribute attr2 = new AttributeImpl(null, "name", _factory.makeTypeName("type"));
    Attribute attr3 = new AttributeImpl(null, "name", _factory.makeTypeName("type1"));
    assertTrue( attr1.equals(attr2) && !attr1.equals(attr3));
  }
  public void test_equals_2() {
    Attribute       commonParent       =       new       AttributeImpl(null,       "parent",
_factory.makeTypeName("type1"));
    Attribute attr1 = new AttributeImpl(commonParent, "name", _factory.makeTypeName("type"));
    Attribute attr2 = new AttributeImpl(commonParent, "name", _factory.makeTypeName("type"));
    Attribute attr3 = new AttributeImpl(null, "name", _factory.makeTypeName("type"));
    assertTrue( attr1.equals(attr2) && !attr1.equals(attr3));
  }
  public void test_equals_3() {
    Attribute parent1 = new AttributeImpl(null, "parent", _factory.makeTypeName("type1"));
    Attribute parent2 = new AttributeImpl(null, "parent", _factory.makeTypeName("type1"));
    Attribute attr1 = new AttributeImpl(parent1, "name", _factory.makeTypeName("type"));
    Attribute attr2 = new AttributeImpl(parent2, "name", _factory.makeTypeName("type"));
    assertTrue( attr1.equals(attr2) );
  }
  public void test_equals_4() {
    Attribute parent1 = _factory.makeAttribute(

_factory.makeAttribute(null,"grandparent",_factory.makeTypeName("grandparent")),
                               "parent", _factory.makeTypeName("type1"));
    Attribute parent2 = _factory.makeAttribute(

_factory.makeAttribute(null,"grandparent",_factory.makeTypeName("grandparent")),
                               "parent", _factory.makeTypeName("type1"));
    Attribute attr1 = _factory.makeAttribute(parent1, "name", _factory.makeTypeName("type"));
    Attribute attr2 = _factory.makeAttribute(parent2, "name", _factory.makeTypeName("type"));
    assertEquals( attr1, attr2 );
  }
  public void test_equals_5() {
    Attribute parent2 = _factory.makeAttribute(

_factory.makeAttribute(null,"grandparent",_factory.makeTypeName("grandparent")),
                               "parent", _factory.makeTypeName("type1"));
    Attribute attr1 = _factory.makeAttribute(null, "name", _factory.makeTypeName("type"));
    Attribute attr2 = _factory.makeAttribute(parent2, "name", _factory.makeTypeName("type"));
    assertTrue( !attr1.equals(attr2) );
  }
  public void test_equals_6() {
    Attribute parent1 = _factory.makeAttribute(

_factory.makeAttribute(null,"grandparent",_factory.makeTypeName("grandparent")),
                              "parentAb", _factory.makeTypeName("type1"));
    Attribute parent2 = _factory.makeAttribute(

_factory.makeAttribute(null,"grandparent",_factory.makeTypeName("grandparent")),
                              "parent", _factory.makeTypeName("type1"));
    Attribute attr1 = _factory.makeAttribute(parent1, "name", _factory.makeTypeName("type"));
    Attribute attr2 = _factory.makeAttribute(parent2, "name", _factory.makeTypeName("type"));
    assertTrue( !attr1.equals(attr2) );
  }
  /**
   * Attributes with different child attributes can be equal.
   */
  public void test_equals_7() {
    Attribute parent1 = _factory.makeAttribute(

_factory.makeAttribute(null,"grandparent",_factory.makeTypeName("grandparent")),
                              "parent", _factory.makeTypeName("type1"));
    Attribute parent2 = _factory.makeAttribute(

_factory.makeAttribute(null,"grandparent",_factory.makeTypeName("grandparent")),
                              "parent", _factory.makeTypeName("type1"));
    Attribute attr1 = _factory.makeAttribute(parent1, "name", _factory.makeTypeName("type"));
    Attribute attr1_1 = _factory.makeAttribute(attr1, "abc", _factory.makeTypeName("type"));
    Attribute attr2 = _factory.makeAttribute(parent2, "name", _factory.makeTypeName("type"));
    Attribute         attr2_1         =         _factory.makeAttribute(attr2,         "name2",
_factory.makeTypeName("type2"));
    assertTrue( !attr1_1.equals(attr2_1) && attr1.equals(attr2) );
  }
  public void test_fullpath_1() {
    FIOMFactory f = _factory;
    Attribute p1 = f.makeAttribute( null, "Parent1", f.makeTypeName("t"));
    Attribute p2 = f.makeAttribute( p1, "Parent2", f.makeTypeName("t"));
    assertEquals( "Parent1", p1.fullPath() );
  }
  public void test_fullpath_2() {
    FIOMFactory f = _factory;
    Attribute p1 = f.makeAttribute( null, "Parent1", f.makeTypeName("t"));
    Attribute p2 = f.makeAttribute( p1, "Parent2", f.makeTypeName("t"));
    assertEquals( "Parent1.Parent2", p2.fullPath() );
  }

  public void test_ancestors_1() {
    FIOMFactory f = _factory;
    Attribute[] a = new Attribute[3];
    a[0] = f.makeAttribute( null, "Attr0", f.makeTypeName("t"));
    a[1] = f.makeAttribute( a[0], "Attr1", f.makeTypeName("t"));
    a[2] = f.makeAttribute( a[1], "Attr2", f.makeTypeName("t"));

    assertTrue( Arrays.equals( a, a[2].pathToArray() ) &&
                a[0].pathToArray().length == 1 );
  }
} //public class ZzzTest_AttributeImpl extends TestCase
```

15. ZzzTest_CCRImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.*;
import java.io.*;
import junit.framework.TestCase;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

public class ZzzTest_CCRImpl extends TestCase
{

  FIOMFactory fiomFactory = FactoryImpl.newFIOMFactory();

    public ZzzTest_CCRImpl(String Name_)
    {
        super(Name_);
    } //public ZzzTest_CCRImpl(String Name_)



    protected void setUp() throws Exception
    {
    } //protected void setUp()

    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_CCRImpl.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)



  /**
   * Tests the constructor and get/set FCR and CCRName.
   */
  public void test_constructor_getset_FCR_CCRName() {
    FCR fcr = new FCRImpl();
    CCR ccr = new CCRImpl( fcr, "System", "The CCR name" );
    assertTrue( ccr.getFCR() == fcr &&
                ccr.getCCRName().equals( "The CCR name" ) );
  }

  public void test_getset_CCRSchema_1() {
    CCRImpl ccr = new CCRImpl( null, "System", "The Name");
    CCRSchema newCcrSchema = new CCRSchemaImpl();
    ccr.setCCRSchema(newCcrSchema);
    assertTrue( ccr.getCCRSchema() == newCcrSchema );
  }
  public void test_getset_javaClassName_xmlNameSpaceURI_1() {
    CCR ccr = new CCRImpl( null, "System", "The Name");
    ccr.setJavaClassName( "a.c.d.f" );
    ccr.setXMLNameSpaceURI( "http://abc.com/xsd/abc.xsd" );
    assertTrue( "a.c.d.f".equals(ccr.getJavaClassName()) &&
                "http://abc.com/xsd/abc.xsd".equals(ccr.getXMLNameSpaceURI()) );
  }
  public void test_getset_SystemName_1() {
    CCR ccr = new CCRImpl( null, "SystemName", "The Name");
    ccr.setJavaClassName( "a.c.d.f" );
    ccr.setXMLNameSpaceURI( "http://abc.com/xsd/abc.xsd" );
    assertTrue( "SystemName".equals(ccr.getSystemName()) &&
                "The Name".equals(ccr.getCCRName()) );
  }
  public void test_isMandatoryFCRAttribute_1() {
    FIOMFactory f = fiomFactory;
    CCRImpl ccr = new CCRImpl( null, "SystemName", "The Name");
    CCRSchema schema = ccr.getCCRSchema();
    Attribute attr1 = schema.addAttribute( "nameAA", fiomFactory.makeTypeName("theType") );
    attr1.setMinOccurs(1);
    Attribute attr2 = schema.addAttribute( "nameA1", fiomFactory.makeTypeName("theType") );

    Attribute[] fcrAttrs1
                  = { f.makeAttribute( null, "FCR_nameAA_1", f.makeTypeName("theType") ),
                     f.makeAttribute( null, "FCR_nameAA_2", f.makeTypeName("theType") )
                   };
    ccr.addFCRToCCRAttributeMapping(
              fcrAttrs1,
              f.makeAttribute(null, "nameAA", fiomFactory.makeTypeName("theType")) );
    // "duplicate" mandatory attributes to ensure the count is correct
    Attribute[] fcrAttrs1a
                  = { f.makeAttribute( null, "FCR_nameAA_1", f.makeTypeName("theType") ),
                     f.makeAttribute( null, "FCR_nameAA_2", f.makeTypeName("theType") )
                   };
    ccr.addFCRToCCRAttributeMapping(
              fcrAttrs1a,
              f.makeAttribute(null, "nameAA", fiomFactory.makeTypeName("theType")) );

    Attribute[] fcrAttrs2
                  = { f.makeAttribute( null, "FCR_nameAA_1_2", f.makeTypeName("theType") ),
                     f.makeAttribute( null, "FCR_nameAA_2_2", f.makeTypeName("theType") )
                   };
    ccr.addFCRToCCRAttributeMapping( fcrAttrs2, attr2.copy() );

    Attribute a = f.makeAttribute(null, fcrAttrs1[0].getName(),
                                      f.makeTypeName(fcrAttrs1[0].getType().toString()));
    Attribute b = f.makeAttribute(null, fcrAttrs1[1].getName(),
                                      f.makeTypeName(fcrAttrs1[1].getType().toString()));
    assertTrue( ccr.mandatoryFCRAttributeCount()==2 &&
                ccr.isMandatoryFCRAttribute(a) &&
                ccr.isMandatoryFCRAttribute(b) &&
                !ccr.isMandatoryFCRAttribute(fcrAttrs2[0].copy()) &&
                !ccr.isMandatoryFCRAttribute(fcrAttrs2[1].copy())
               );
  }
  public void test_isMandatoryFCRAttribute_2() {
    CCRImpl ccr = new CCRImpl( null, "SystemName", "The Name");
    assertTrue(      !ccr.isMandatoryFCRAttribute(fiomFactory.makeAttribute(null,      "name",
fiomFactory.makeTypeName("theType")) ));
  }
  public void test_compareTo_1() {
    CCRImpl ccr1 = new CCRImpl( null, "SystemName", "The Name");
    CCRImpl ccr2 = new CCRImpl( null, "SystemName2", "1");
    assertTrue( ccr1.compareTo(ccr2)<0 );
  }
  public void test_compareTo_2() {
    CCRImpl ccr1 = new CCRImpl( null, "SystemName", "The Name");
    CCRImpl ccr2 = new CCRImpl( null, "SystemName", "The Name");
    assertTrue( ccr1.compareTo(ccr2)==0 );
  }
  public void test_compareTo_3() {
    CCRImpl ccr1 = new CCRImpl( null, "SystemName", "The Name");
    CCRImpl ccr2 = new CCRImpl( null, "A", "The Name");
    assertTrue( ccr1.compareTo(ccr2)>0 );
  }
  public void test_compareTo_4() {
    CCRImpl ccr1 = new CCRImpl( null, "SystemName", "The Name");
    CCRImpl ccr2 = new CCRImpl( null, "SystemName", "A");
    assertTrue( ccr1.compareTo(ccr2)>0 );
  }
  public void test_compareTo_5() {
    CCRImpl ccr1 = new CCRImpl( null, "SystemName", "The Name");
    CCRImpl ccr2 = new CCRImpl( null, "SystemName", "X");
    assertTrue( ccr1.compareTo(ccr2)<0 );
  }

  public void test_joinCCR_1() {
    CCRImpl ccr = (CCRImpl) fiomFactory.makeCCR( null, "S", "T_CCR" );
    FCRImpl fcr = new FCRImpl( null, "FCR" );
    ccr.joinFCR(fcr);
    assertTrue( ccr.getFCR()==fcr && fcr.findCCR("S","T_CCR")==ccr );
  }

  public static void p(String m) {
    System.out.println(m);
  }
} //public class ZzzTest_CCRImpl extends TestCase
```

16. ZzzTest_FactoryImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import junit.framework.TestCase;
import mil.navy.nps.cs.oomi.fiom.*;

public class ZzzTest_FactoryImpl extends TestCase
{



    public ZzzTest_FactoryImpl(String Name_)
    {
        super(Name_);
    } //public ZzzTest_FactoryImpl(String Name_)

    protected void setUp()
    {
    } //protected void setUp()

    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_FactoryImpl.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)

  protected static void p(String m) {
    System.out.println(m);
  }

  public void test_0() {
    try {
      FIOMFactory _factory = FactoryImpl.newFIOMFactory();
      assertTrue(_factory!=null);
    }
    catch (Exception exc) {
      exc.printStackTrace();
      fail(exc.getClass() + ": " + exc.getMessage());
    }

  }
  FIOMFactory _factory = FactoryImpl.newFIOMFactory();

  public void test_1() {
    TypeName type = _factory.makeTypeName( "Type1" );
    Attribute attr = _factory.makeAttribute( null, "Name", type );
    assertTrue(attr.getName().equals("Name") && attr.getType().toString().equals("Type1"));
  }

} //public class ZzzTest_FactoryImpl extends TestCase
```

17. ZzzTest_FCRImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.*;
import java.io.*;
import java.net.URL;
import junit.framework.TestCase;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

public class ZzzTest_FCRImpl extends TestCase
{

    public ZzzTest_FCRImpl(String Name_)
    {
        super(Name_);
    } //public ZzzTest_FCRImpl(String Name_)

    FIOM    fiom;
    FE      fe_1;

    String fev_1_name = "FE 1. FEV 1";
    FEVImpl fevImpl_1;
    FEV     fev_1;

    String fcr_1_name = "FE 1. FEV 1. FCR 1";
    FCRImpl fcrImpl_1;
    FCR     fcr_1;

    protected void setUp() throws Exception
    {
      fiom = new FIOMImpl("a FIOM");
      fe_1   = fiom.newFE( "FE 1" );
      fevImpl_1 = (FEVImpl)fe_1.createRootFEV(fev_1_name);
      fev_1 = fevImpl_1;
      fcrImpl_1 = new FCRImpl(fev_1, "FCR Impl 1");
      fcr_1 = fcrImpl_1;
      //fcrImpl_1 = (FCRImpl) fev_1.newF
    } //protected void setUp()
```

```
  protected void tearDown()
  {
  } //protected void tearDown()

  public static void main(String[] args)
  {
      String[] testCaseName = {ZzzTest_FCRImpl.class.getName()};
      junit.swingui.TestRunner.main(testCaseName);
  } //public static void main(String[] args)

public void test_constructor_getsetFEV_1() {
  FCR fcr = new FCRImpl(fev_1, "The Name");
  assertTrue( fcr.getFEV()==fev_1 );
}
public void test_constructor_getsetFCRName_1() {
  FCR fcr = new FCRImpl(fev_1, "The Name");
  assertEquals( "The Name", fcr.getFCRName() );
}
public void test_ccrCount_0() {
  FCR fcr = new FCRImpl(fev_1, "The Name");
  assertEquals( 0, fcr.ccrCount() );
}

public void test_constructor_getset_JavaClassName() {
  FCR fcr = new FCRImpl();
  fcr.setJavaClassName("a.b.C");
  assertEquals( "a.b.C", fcr.getJavaClassName() );
}

public void test_getaddCCR_ccrCount_1() {
  CCR[] ccrs = new CCR[] { new CCRImpl(null, "System", "CCR 0"),
                           new CCRImpl(null, "System", "CCR 1") };
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  fcr.addCCR(ccrs[0]);
  fcr.addCCR(ccrs[1]);
  CCR[] ccrSorted = fcr.getCCR();
  Arrays.sort( ccrSorted );
  assertTrue( fcr.ccrCount()==ccrs.length &&
              Arrays.equals(ccrs, ccrSorted) &&
              ccrs[0].getFCR()==fcr &&
              ccrs[1].getFCR()==fcr  );
}
/**
```

```
 * Call setCCR() twice, to ensure the array replaces the existing and not
 * add to it.
 */
public void test_getaddCCR_ccrCount_2() {
  CCR[] ccrs = new CCR[] { new CCRImpl(null, "System",  "CCR 0"),
                           new CCRImpl(null, "System", "CCR 1") };
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  // call setCCR() twice, to ensure
  fcr.addCCR(ccrs[0]);
  fcr.addCCR(ccrs[1]);
  fcr.addCCR(ccrs[0]);
  fcr.addCCR(ccrs[1]);
  assertTrue( fcr.ccrCount()==ccrs.length &&
              Arrays.equals(ccrs, fcr.getCCR()) &&
              ccrs[0].getFCR()==fcr &&
              ccrs[1].getFCR()==fcr  );
}
public void test_findCCR_1() {
  CCR[] ccrs = new CCR[] { new CCRImpl(null, "System", "CCR 0"),
                           new CCRImpl(null, "System", "CCR 1") };
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  fcr.addCCR(ccrs[0]);
  fcr.addCCR(ccrs[1]);
  assertTrue( fcr.findCCR("System", "ccr 0")==ccrs[0] &&
              fcr.findCCR("System", "ccr 1")==ccrs[1] );
}
public void test_findCCR_2_null() {
  CCR[] ccrs = new CCR[] { new CCRImpl(null, "System", "CCR 0"),
                           new CCRImpl(null, "System", "CCR 1") };
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  fcr.addCCR(ccrs[0]);
  fcr.addCCR(ccrs[1]);
  assertTrue( fcr.findCCR("System", "ccr 0234")==null );
}
public void test_findCCR_bySystem_1() {
  CCR[] ccrs = new CCR[] { new CCRImpl(null, "System A", "CCR A 0"),
                           new CCRImpl(null, "System", "CCR 1"),
                           new CCRImpl(null, "System B", "CCR 1"),
                           new CCRImpl(null, "System a", "CCR A 1")
                           };
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  fcr.addCCR(ccrs[0]);
  fcr.addCCR(ccrs[1]);
  fcr.addCCR(ccrs[2]);
  fcr.addCCR(ccrs[3]);
  CCR[] ccrsFound = fcr.findCCR( "system a" );
  Arrays.sort(ccrsFound);
  try{
  Class c = Class.forName("java.lang.String");
  String s =(String) c.newInstance();

  }
  catch (Exception e) {}
  assertTrue( ccrsFound.length==2 &&
              ccrsFound[0].getCCRName().equals("CCR A 0") &&
              ccrsFound[1].getCCRName().equals("CCR A 1") );
}
public void test_ccrExists_true() {
  CCR[] ccrs = new CCR[] { new CCRImpl(null, "System", "CCR 0"),
                           new CCRImpl(null, "System", "CCR 1") };
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  fcr.addCCR(ccrs[0]);
  fcr.addCCR(ccrs[1]);
  assertTrue( fcr.ccrExists("System", "ccR 0") &&
              fcr.ccrExists("System", "CCR 1") );
}
public void test_ccrExists_false() {
  CCR[] ccrs = new CCR[] { new CCRImpl(null, "System", "CCR 0"),
                           new CCRImpl(null, "System", "CCR 1") };
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  fcr.addCCR(ccrs[0]);
  fcr.addCCR(ccrs[1]);
  assertTrue( !fcr.ccrExists("System", "A ccR 0") );
}
/**
 * Same SystemName, different CCRName.
 */
public void test_newCCR_1_1() throws Exception {
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  CCR[] ccrs = new CCR[] { fcr.newCCR("System", "1", null),
                           fcr.newCCR("System", "2", null) };
  assertTrue( fcr.ccrCount()==ccrs.length  );
}
/**
 * Same SystemName, different CCRName.
 */
```

```
public void test_newCCR_1_2() throws Exception {
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  CCR[] ccrs = new CCR[] { fcr.newCCR("System", "1", null),
                           fcr.newCCR("System", "2", null) };
  CCR[] ccrSorted = fcr.getCCR();
  Arrays.sort( ccrSorted );
  assertTrue( Arrays.equals(ccrs, ccrSorted) );
}
/**
 * Same SystemName, different CCRName.
 */
public void test_newCCR_1_3() throws Exception {
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  CCR[] ccrs = new CCR[] { fcr.newCCR("System", "1", null),
                           fcr.newCCR("System", "2", null) };
  assertTrue(
              ccrs[0].getFCR()==fcr &&
              ccrs[1].getFCR()==fcr  );
}
/**
 * Same CCRName, different SystemName.
 */
public void test_newCCR_2() throws Exception {
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  CCR[] ccrs = new CCR[] { fcr.newCCR("System", "1", null),
                           fcr.newCCR("System", "2", null) };
  CCR[] ccrSorted = fcr.getCCR();
  Arrays.sort( ccrSorted );
  assertTrue( fcr.ccrCount()==ccrs.length &&
              Arrays.equals(ccrs, ccrSorted) &&
              ccrs[0].getFCR()==fcr &&
              ccrs[1].getFCR()==fcr  );
}
/**
 * newCCR() with duplicate CCRName (case-insensitive) and SystemName.
 */
public void test_newCCR_duplicate_ccr_1() {
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  try {
    fcr.newCCR("System", "1", null);
    fcr.newCCR("System", "2 abcdef", null);
    fcr.newCCR("System", "2 AbcDef", null);
    fail(DuplicateKeyException.class.getName() + " expected but not thrown.");
  }
  catch (Exception exc) {
    assertEquals( DuplicateKeyException.class, exc.getClass() );
  }
}
/**
 * newCCR() with duplicate CCRName and SystemName (case-insensitive).
 */
public void test_newCCR_duplicate_ccr_2() {
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  try {
    fcr.newCCR("System", "1", null);
    fcr.newCCR("System", "2 abcdef", null);
    fcr.newCCR("SyStem", "2 abcDef", null);
    fail(DuplicateKeyException.class.getName() + " expected but not thrown.");
  }
  catch (Exception exc) {
    assertEquals( DuplicateKeyException.class, exc.getClass() );
  }
}
/**
 * Duplicate SystemName and xmlNameSpaceURI
 * should result in DuplicateKeyException.
 */
public void test_newCCR_duplicate_url_1() {
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  try {
    String urlString = "http://q.w.com/abc/dfg/a.xsd";
    fcr.newCCR("System", "1", null);
    fcr.newCCR("System", "2 abcdef", new URL(urlString));
    fcr.newCCR("System", "3 AbcDef", new URL(urlString));
    fail(DuplicateKeyException.class.getName() + " expected but not thrown.");
  }
  catch (Exception exc) {
    assertEquals( DuplicateKeyException.class, exc.getClass() );
  }
}
public void test_getset_FCRSchema_1() throws Exception {
  FCRImpl fcr = new FCRImpl(fev_1, "The Name");
  FCRSchema newFcrSchema = new FCRSchemaImpl();
  fcr.setFCRSchema(newFcrSchema);
  assertTrue( fcr.getFCRSchema() == newFcrSchema );
}
```

```
  public void test_findCCR_by_xmlSchemaXML_1() throws Exception  {
    String urlString = "http://a.b.com/xsd2.xsd";
    FCRImpl fcr = new FCRImpl(fev_1, "The Name");
    CCR[] ccrs = new CCR[] { fcr.newCCR("System", "1", new URL("http://a.b.com/xsd1.xsd")),
                             fcr.newCCR("System", "2", new URL(urlString))
                            };
    assertTrue( ccrs[1] == fcr.findCCR("SysTEM", new URL(urlString)) );
  }
  /**
   * Tests case-sensitive findCCR(URL).
   */
  public void test_findCCR_by_xmlSchemaXML_2() throws Exception  {
    String urlString = "http://a.b.com/xsd2.xsd";
    FCRImpl fcr = new FCRImpl(fev_1, "The Name");
    CCR[] ccrs = new CCR[] { fcr.newCCR("System", "1", new URL("http://a.b.com/xsd1.xsd")),
                             fcr.newCCR("System", "2", new URL(urlString))
                            };
    assertTrue( null == fcr.findCCR("SYSTEM", new URL(urlString.toUpperCase())) );
  }
  /**
   * Tests findCCR(null).
   */
  public void test_findCCR_by_xmlSchemaXML_3() throws Exception  {
    String urlString = "http://a.b.com/xsd2.xsd";
    FCRImpl fcr = new FCRImpl(fev_1, "The Name");
    CCR[] ccrs = new CCR[] { fcr.newCCR("System", "1", new URL("http://a.b.com/xsd1.xsd")),
                             fcr.newCCR("System", "2", new URL(urlString))
                            };
    assertTrue( null == fcr.findCCR("System", (URL)null) );
  }

} //public class ZzzTest_FCRImpl extends TestCase
```

18. ZzzTest_FEImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.*;
import java.io.*;
import java.net.URL;
import junit.framework.TestCase;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

public class ZzzTest_FEImpl extends TestCase
{

    public ZzzTest_FEImpl(String Name_)
    {
        super(Name_);
    } //public ZzzTest_FEImpl(String Name_)

    // Test variables.
    OOMIDatabase oomiDb = null;
    FIOM fiom = null;
    FEImpl feImpl = null;
    FE fe = null;
    FEV[] fevArray = null;

    protected void setUp() throws Exception
    {
      OOMIDatabase oomiDb = new OOMIDatabaseImpl();
      FIOM fiom = oomiDb.newFIOM("a fiom");
      feImpl = (FEImpl)fiom.newFE("FE 0");
      fe = (FE)feImpl;
      fevArray = new FEVImpl[] { new FEVImpl(null, null, "FEV 0"),
                                 new FEVImpl(null, null, "FEV 1") };
    } //protected void setUp()

    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_FEImpl.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)


  public void test_constructor_1() {
    String feName = "a FE";
    FE fe = new FEImpl(fiom, null, feName);
    assertTrue( fe.getFIOM()==fiom && fe.getFEName().equals(feName) );
```

```
}
/**
 * Tests getRootFEV() and setRootFEV().
 * Calls setRootFEV() once.
 */
public void test_set_getFEV_2() {
  FEV fev = new FEVImpl();
  feImpl.setRootFEV(fev);
  assertTrue( fev == fe.getRootFEV() );
}
/**
 * Tests getRootFEV() and setRootFEV().
 * Calls setRootFEV() twice.
 */
public void test_set_getFEV_3() {
  feImpl.setRootFEV(new FEVImpl());
  FEV fev = new FEVImpl();
  feImpl.setRootFEV(fev);
  assertTrue( fev == feImpl.getRootFEV() );
}
/**
 * Tests getRootFEV() and setRootFEV() to check if FE property is properly set.
 * Calls setRootFEV() once.
 */
public void test_set_getFEV_4() {
  FEV fev = new FEVImpl();
  feImpl.setRootFEV(fev);
  assertTrue( fev.getFE() == fe );
}

public void test_equals_true() {
  FIOM fiom1 = new FIOMImpl("my name");
  FIOM fiom2 = new FIOMImpl("my nAme");
  assertEquals( fiom1, fiom2 );
}
public void test_equals_false() {
  FIOM fiom1 = new FIOMImpl("my name1");
  FIOM fiom2 = new FIOMImpl("my name");
  assertTrue( ! fiom1.equals(fiom2) );
}
public void test_equals_null() {
  FIOM fiom1 = new FIOMImpl("my name1");
  assertTrue( ! fiom1.equals(null) );
```

```
}

public void test_findFEV_1() throws Exception {
  FEV rootFEV = feImpl.createRootFEV("ROOT");
  rootFEV.newChild("child fev 1");
  FEV target = rootFEV.newChild("child fev 2").newChild("2.1");
  assertTrue( fe.findFEV("2.1") == target );
}
/**
 * Test finding the root FEV.
 */
public void test_findFEV_2() throws Exception {
  FEV rootFEV = feImpl.createRootFEV("ROOT");
  rootFEV.newChild("child fev 1");
  FEV target = rootFEV.newChild("child fev 2").newChild("2.1");
  assertTrue( fe.findFEV("ROOT") == rootFEV );
}
public void test_fevExists_1() throws Exception {
  FEV rootFEV = feImpl.createRootFEV("ROOT");
  rootFEV.newChild("child fev 1");
  FEV target = rootFEV.newChild("child fev 2").newChild("2.1");
  assertTrue( fe.fevExists("2.1") );
}
public void test_createFEV_1 () throws Exception {
  FEV rootFEV = fe.createRootFEV("ROOT");
  assertTrue( fe.getRootFEV() == rootFEV );
}
public void test_createFEV_2 () throws Exception {
  FEV rootFEV = fe.createRootFEV("ROOT");
  assertTrue( fe.getRootFEV().getFE() == fe );
}

/**
 * Test get/set Parent.
 */
public void test_getParent_setParent_1() throws Exception {
  FE parent = new FEImpl(  );
  FEImpl fe = new FEImpl(  null, parent,  "abc" );
  assertTrue( fe.getParent() == parent );
}
/**
 * Test newChild() and getChild().
 */
```

```
public void test_newChild_getChild_1() throws Exception {
  FE fe = new FEImpl();
  FE child = fe.newChild("child 1");
  assertTrue( child.getParent()==fe );
}
/**
 * Test newChild() and getChild().
 */
public void test_newChild_getChild_2() throws Exception {
  FE fe = new FEImpl();
  FE[] children = new FE[] { fe.newChild("child 1"),
                             fe.newChild("child 2") };
  assertTrue( Arrays.equals( children, fe.getChild() ) );
}
/**
 * Test newChild() and getChild().
 */
public void test_newChild_getChild_3() throws Exception {
  FE fe = new FEImpl();
  FE[] children = new FE[] { fe.newChild("child 1"),
                             fe.newChild("child 2") };
  assertTrue( children[0].getFEName().equals("child 1") &&
              children[1].getFEName().equals("child 2") );
}
/**
 * Test setChild() Parent property set.
 */
public void test_setChild_Parent() throws Exception {
  FEImpl fe = new FEImpl();
  FE[] children = new FE[] { new FEImpl(null, null, "child 1"),
                             new FEImpl(null, null, "child 2") };
  fe.setChild(children);
  FE[] ch = fe.getChild();
  assertTrue( ch[0].getParent()==fe &&
              ch[1].getParent()==fe );
}
/**
 * Test setChild() FIOM property set.
 */
public void test_setChild_FIOM() throws Exception {
  FEImpl fe = new FEImpl();
  fe.setFIOM(new FIOMImpl());
  FE[] children = new FE[] { new FEImpl(null, null, "child 1"),
                              new FEImpl(null, null, "child 2") };
  fe.setChild(children);
  FE[] ch = fe.getChild();
  assertTrue( ch[0].getFIOM()==fe.getFIOM() &&
              ch[1].getFIOM()==fe.getFIOM() );
}

/**
 * Test childCount() and that it counts only the first level.
 */
public void test_childCount_1() throws Exception {
  FE fe = new FEImpl();
  fe.newChild("child 1");
  assertEquals( 1, fe.childCount() );
}
/**
 * Test childCount() and that it counts only the first level.
 */
public void test_childCount_2() throws Exception {
  FE fe = new FEImpl();
  FE[] children = new FE[] { fe.newChild("child 1"),
                              fe.newChild("child 2") };
  children[0].newChild("random child");
  assertEquals( children.length, fe.childCount() );
}
/**
 * Recursive findDescendent()
 */
public void test_findDescendent_A() throws Exception {
  FE fe = new FEImpl();
  FE target = fe.newChild("child 1");

  assertTrue( fe.findDescendent("CHild 1", true) == target );
}
/**
 * Recursive findDescendent()
 */
public void test_findDescendent_B() throws Exception {
  FE fe = new FEImpl();
  FE target = fe.newChild("child 1");
  fe.newChild("child 2");

  assertTrue( fe.findDescendent("CHild 1", true) == target );
```

```
}
/**
 * Recursive findDescendent()
 */
public void test_findDescendent_C() throws Exception {
  FE fe = new FEImpl();
  fe.newChild("child 1");
  FE target = fe.newChild("child 2");

  assertTrue( fe.findDescendent("CHild 2", true) == target );
}
/**
 * Non-Recursive findDescendent()
 */
public void test_findDescendent_D() throws Exception {
  FE fe = new FEImpl();
  fe.newChild("child 1");
  FE target = fe.newChild("child 2");

  assertTrue( fe.findDescendent("CHild 2", false) == target );
}
/**
 * Recursive findDescendent()
 */
public void test_findDescendent_1() throws Exception {
  FE fe = new FEImpl();
  FE[] children = new FE[] { fe.newChild("child 1"),
                             fe.newChild("child 2") };
  FE target = children[0].newChild("child 11");
  target.newChild("child 111");
  children[0].newChild("child 12");

  assertTrue( fe.findDescendent("CHild 11", true) == target );
}
/**
 * Non-recursive findDescendent()
 */
public void test_findDescendent_2() throws Exception {
  FE fe = new FEImpl();
  FE[] children = new FE[] { fe.newChild("child 1"),
                             fe.newChild("child 2") };
  FE target = children[0].newChild("child 11");
  target.newChild("child 111");
  children[0].newChild("child 12");

  assertTrue( ! (fe.findDescendent("child 111", false) == target) );
}
/**
 * Recursive findDescendent(), child found at first level.
 */
public void test_findDescendent_3() throws Exception {
  FE fe = new FEImpl();
  FE[] children = new FE[] { fe.newChild("child 1"),
                             fe.newChild("child 2") };
  FE target = children[0].newChild("child 11");
  target.newChild("child 111");
  children[0].newChild("child 12");

  assertTrue( fe.findDescendent("CHild 2", true) == children[1] );
}
/**
 * Recursive hasDescendent()
 */
public void test_hasDescendent_1() throws Exception {
  FEImpl fe = new FEImpl();
  FE[] children = new FE[] { fe.newChild("child 1"),
                             fe.newChild("child 2") };
  FE target = children[0].newChild("child 11");
  target.newChild("child 111");
  children[0].newChild("child 12");

  assertTrue( fe.hasDescendent("CHild 111", true) );
}
/**
 * Non-recursive hasDescendent()
 */
public void test_hasDescendent_2() throws Exception {
  FEImpl fe = new FEImpl();
  FE[] children = new FE[] { fe.newChild("child 1"),
                             fe.newChild("child 2") };
  FE target = children[0].newChild("child 11");
  target.newChild("child 111");
  children[0].newChild("child 12");

  assertTrue( ! fe.hasDescendent("child 111", false) );
}
```

```
/**
 * Test newChild with duplicate name.
 */
public void test_newChild_duplicate_1() {
  try {
    FE fe = new FEImpl();
    FE[] children = new FE[] { fe.newChild("child 1"),
                             fe.newChild("child 2") };
    FE target = children[0].newChild("child 11");
    FE c = target.newChild("child 111");
    fe.newChild(c.getFEName());
    fail(DuplicateKeyException.class.getName() + " expected, but not thrown" );
  }
  catch (Exception exc) {
    assertEquals(  DuplicateKeyException.class, exc.getClass() );
  }
}
/**
 * findCCR(), target in this FE's child's list of CCRs.
 */
public void test_findCCR_1() throws Exception {
  String systemName = "System to be found";
  String ccrName = "CCR Name to be found";
  String urlString = "http://findme.com/found.xsd";

  FE rootFE = new FEImpl();
  FE fe;
  FCR fcr;

  // level 1
  String suffix = "0";
  fe = rootFE.newChild( "Child" + suffix );
    fcr = fe.createRootFEV( "FEV" + suffix ).createFCR( "FCR" + suffix );
  CCR target = fcr.newCCR( systemName, ccrName, new URL(urlString) );

  assertTrue(
    rootFE.findCCR(systemName, new URL(urlString)) == target );
}
/**
 * findCCR(), target in this FE's own list of CCRs.
 */
public void test_findCCR_2() throws Exception {
  String systemName = "System to be found";
  String ccrName = "CCR Name to be found";
  String urlString = "http://findme.com/found.xsd";

  FE rootFE = new FEImpl();
    FCR rootFCR = rootFE.createRootFEV("ROOT").createFCR("ROOT FCR");
    rootFCR.newCCR("s", "cc", new URL("http://a.com/asdf.xsd"));
  FE fe;
  FCR fcr;

  // level 1
  String suffix = "0";
  fe = rootFE.newChild( "Child" + suffix );
    fcr = fe.createRootFEV( "FEV" + suffix ).createFCR( "FCR" + suffix );
    fcr.newCCR( "System" + suffix, "CCR" + suffix,
                new URL("http://a.com/0.xsd" + suffix) );

  CCR target = rootFCR.newCCR(systemName, ccrName, new URL(urlString));

  assertTrue(
    rootFE.findCCR(systemName, new URL(urlString)) == target &&
    rootFE.findDescendentCCR(systemName, new URL(urlString)) == null );
}
/**
 * findCCR(String systemName).
 */
public void test_findCCR_3() throws Exception {
  String systemName = "System to be found";
  String ccrName = "CCR Name to be found";
  String urlString = "http://findme.com/found.xsd";

  FE rootFE = new FEImpl();
    FCR rootFCR = rootFE.createRootFEV("ROOT").createFCR("ROOT FCR");
    rootFCR.newCCR("s", "cc", new URL("http://a.com/asdf.xsd"));
  FE fe;
  FCR fcr;

  // level 1
  String suffix = "0";
  fe = rootFE.newChild( "Child" + suffix );
    fcr = fe.createRootFEV( "FEV" + suffix ).createFCR( "FCR" + suffix );
    CCR ccr1 = fcr.newCCR( "System" + suffix, "CCR 1" + suffix,
                           new URL("http://a.com/0.xsd" + suffix) );
    fcr.newCCR( "SystemA" + suffix, "CCR" + suffix,
                new URL("http://a.com/0.xsd" + suffix) );
    CCR ccr2 = fcr.newCCR( "System" + suffix, "CCR 2" + suffix,
                            new URL("http://a.b.com/0.xsd" + suffix) );

  CCR target = rootFCR.newCCR(systemName, ccrName, new URL(urlString));
  CCR[] result = fe.findCCR(ccr1.getSystemName());
  Arrays.sort( result );
  assertTrue( result[0].getCCRName().equals(ccr1.getCCRName()) &&
              result[1].getCCRName().equals(ccr2.getCCRName())
     );
}
/**
 * findDescendentCCR(), target in first-level.
 */
public void test_findDescendentCCR_1() throws Exception {
  String systemName = "System to be found";
  String ccrName = "CCR Name to be found";
  String urlString = "http://findme.com/found.xsd";

  FE rootFE = new FEImpl();
  FE fe;
  FCR fcr;

  // level 1
  String suffix = "0";
  fe = rootFE.newChild( "Child" + suffix );
    fcr = fe.createRootFEV( "FEV" + suffix ).createFCR( "FCR" + suffix );
  CCR target = fcr.newCCR( systemName, ccrName, new URL(urlString) );

  assertTrue(
    rootFE.findDescendentCCR(systemName, new URL(urlString)) == target );
}

/**
 * Recursive findDescendentCCR(), target in grandchild
 */
public void test_findDescendentCCR_2() throws Exception {
  String systemName = "System to be found";
  String ccrName = "CCR Name to be found";
  String urlString = "http://findme.com/found.xsd";

  FE rootFE = new FEImpl();
```

```
    FE fe;
    FCR fcr;
    // level 1
    String suffix = "0";
    fe = rootFE.newChild( "Child" + suffix );
      fcr = fe.createRootFEV( "FEV" + suffix ).createFCR( "FCR" + suffix );
      fcr.newCCR( "System" + suffix, "CCR" + suffix,
                  new URL("http://a.com/0.xsd" + suffix) );
    // level 2
    suffix = "0.1";
    fe = fe.newChild( "Child" + suffix );
      fcr = fe.createRootFEV( "FEV" + suffix ).createFCR( "FCR" + suffix );
      fcr.newCCR( "System" + suffix, "CCR" + suffix,
                  new URL("http://a.com/0.xsd" + suffix) );
    // level 1
    suffix = "1";
    fe = rootFE.newChild( "Child" + suffix );
      fcr = fe.createRootFEV( "FEV" + suffix ).createFCR( "FCR" + suffix );
      fcr.newCCR( "System" + suffix, "CCR" + suffix,
                  new URL("http://a.com/0.xsd" + suffix) );
    // level 2
    suffix = "1.1";
    fe = fe.newChild( "Child" + suffix );
      fcr = fe.createRootFEV( "FEV" + suffix ).createFCR( "FCR" + suffix );
      fcr.newCCR( "System" + suffix, "CCR" + suffix,
                  new URL("http://a.com/0.xsd" + suffix) );

    CCR target = fcr.newCCR( systemName, ccrName, new URL(urlString) );

    assertTrue( rootFE.findDescendentCCR(systemName, new URL(urlString))
                == target );
  }


} //public class ZzzTest_FEImpl extends TestCase
```

19. ZzzTest_FEVImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.*;
import java.io.*;
import java.net.URL;
import junit.framework.TestCase;

import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

public class ZzzTest_FEVImpl extends TestCase
{

    public ZzzTest_FEVImpl(String Name_)
    {
        super(Name_);
    } //public ZzzTest_FEVImpl(String Name_)



    FIOM fiom;
    FE   fe_1;

    protected void setUp() throws Exception
    {
      fiom = new FIOMImpl("a FIOM");
      fe_1   = fiom.newFE( "FE 1" );
    } //protected void setUp()s

    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_FEVImpl.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)

  /**
   * Test FEVImpl(FE, String), get/set Parent, FE and FEVName.
   */
  public void test_constructor_1() {
    FEV parent = new FEVImpl( null, fe_1, "root FEV");
    FEV fev = new FEVImpl( parent, fe_1, "this FEV" );

    assertTrue ( fev.getFEVName().equals("this FEV") &&
                 fev.getFE() == fe_1 &&
                 fev.getParent() == parent );
  }
  public void test_equals_true() {
    FEV fev1 = new FEVImpl(null, fe_1, "my namE");
    FEV fev2 = new FEVImpl(null, fe_1, "my name");
    assertTrue( fev1.equals(fev2) && fev1.hashCode()==fev2.hashCode() );
  }
  public void test_equals_false_name_not_equal() {
    FEV fev1 = new FEVImpl(null, fe_1, "my  name");
    FEV fev2 = new FEVImpl(null, fe_1, "my name");
    assertTrue( ! fev1.equals(fev2) );
  }
  public void test_equals_false_fe_not_equal() {
    FEV fev1 = new FEVImpl(null, fe_1, "my name");
    FEV fev2 = new FEVImpl(null, new FEImpl(), "my name");
    assertTrue( ! fev1.equals(fev2) );
  }
  public void test_equals_null() {
    FEV fev1 = new FEVImpl(null, fe_1, "my name");
    assertTrue( ! fev1.equals(null) );
  }

  public void test_setgetFCR_1() {
    FEVImpl fevImpl1 = new FEVImpl(null, fe_1, "my name");
    FCR fcr = new FCRImpl();
    fevImpl1.setFCR(fcr);
    assertTrue( fevImpl1.getFCR()==fcr );
  }
  public void test_setgetFCR_2() {
    FEVImpl fevImpl1 = new FEVImpl(null, fe_1, "my name");
    FCR fcr = new FCRImpl();
    fevImpl1.setFCR(fcr);
    assertTrue( fevImpl1.getFCR().getFEV()==fevImpl1 );
  }
  public void test_create_1() {
    FEV fev = new FEVImpl();
    FCR fcr = fev.createFCR("FCR name 1");
    assertTrue( fev.getFCR()==fcr );
  }
  /**
   * Calls createFCR() twice, to ensure first FCR value is replaced.
   */
  public void test_create_2() {
    FEV fev = new FEVImpl();
    FCR fcr1 = fev.createFCR("FCR name 1");
    FCR fcr2 = fev.createFCR("FCR name 2");
    assertTrue( fev.getFCR()==fcr2 );
  }

  /**
   * Test get/set Parent.
   */
  public void test_getParent_setParent_1() throws Exception {
    FEV parent = new FEVImpl(  );
    FEVImpl fev = new FEVImpl(  parent, null, "abc" );
    assertTrue( fev.getParent() == parent );
  }
  /**
   * Test newChild() and getChild().
   */
  public void test_newChild_getChild_1() throws Exception {
    FEV fev = new FEVImpl();
    FEV child = fev.newChild("child 1");
    assertTrue( child.getParent()==fev );
  }
  /**
   * Test newChild() and getChild().
   */
  public void test_newChild_getChild_2() throws Exception {
    FEV fev = new FEVImpl();
    FEV[] children = new FEV[] { fev.newChild("child 1"),
                                 fev.newChild("child 2") };
    assertTrue( Arrays.equals( children, fev.getChild() ) );
  }
  /**
   * Test newChild() and getChild().
   */
  public void test_newChild_getChild_3() throws Exception {
    FEV fev = new FEVImpl();
    FEV[] children = new FEV[] { fev.newChild("child 1"),
                                 fev.newChild("child 2") };
    assertTrue( children[0].getFEVName().equals("child 1") &&
                children[1].getFEVName().equals("child 2") );
  }
  /**
   * Test setChild() Parent property set.
   */
  public void test_setChild_Parent() throws Exception {
    FEVImpl fev = new FEVImpl();
    FEV[] children = new FEV[] { new FEVImpl(null, null, "child 1"),
                                 new FEVImpl(null, null, "child 2") };
    fev.setChild(children);
    FEV[] ch = fev.getChild();
    assertTrue( ch[0].getParent()==fev &&
                ch[1].getParent()==fev );
  }
  /**
   * Test setChild() FE property set.
   */
  public void test_setChild_FE() throws Exception {
    FEVImpl fev = new FEVImpl();
    fev.setFE(new FEImpl());
    FEV[] children = new FEV[] { new FEVImpl(null, null, "child 1"),
                                 new FEVImpl(null, null, "child 2") };
    fev.setChild(children);
    FEV[] ch = fev.getChild();
    assertTrue( ch[0].getFE()==fev.getFE() &&
                ch[1].getFE()==fev.getFE() );
  }

  /**
   * Test childCount() and that it counts only the first level.
   */
  public void test_childCount_1() throws Exception {
    FEV fev = new FEVImpl();
    fev.newChild("child 1");
    assertEquals( 1, fev.childCount() );
  }
  /**
   * Test childCount() and that it counts only the first level.
   */
  public void test_childCount_2() throws Exception {
    FEV fev = new FEVImpl();
    FEV[] children = new FEV[] { fev.newChild("child 1"),
                                 fev.newChild("child 2") };
    children[0].newChild("random child");
    assertEquals( children.length, fev.childCount() );
  }
  /**
```

```
 * Recursive findDescendent()
 */
public void test_findDescendent_A() throws Exception {
  FEV fev = new FEVImpl();
  FEV target = fev.newChild("child 1");

  assertTrue( fev.findDescendent("CHild 1", true) == target );
}
/**
 * Recursive findDescendent()
 */
public void test_findDescendent_B() throws Exception {
  FEV fev = new FEVImpl();
  FEV target = fev.newChild("child 1");
  fev.newChild("child 2");

  assertTrue( fev.findDescendent("CHild 1", true) == target );
}
/**
 * Recursive findDescendent()
 */
public void test_findDescendent_C() throws Exception {
  FEV fev = new FEVImpl();
  fev.newChild("child 1");
  FEV target = fev.newChild("child 2");

  assertTrue( fev.findDescendent("CHild 2", true) == target );
}
/**
 * Non-Recursive findDescendent()
 */
public void test_findDescendent_D() throws Exception {
  FEV fev = new FEVImpl();
  fev.newChild("child 1");
  FEV target = fev.newChild("child 2");

  assertTrue( fev.findDescendent("CHild 2", false) == target );
}
/**
 * Recursive findDescendent()
 */
public void test_findDescendent_1() throws Exception {
  FEV fev = new FEVImpl();
  FEV[] children = new FEV[] { fev.newChild("child 1"),
                               fev.newChild("child 2") };
  FEV target = children[0].newChild("child 11");
  target.newChild("child 111");
  children[0].newChild("child 12");

  assertTrue( fev.findDescendent("CHild 11", true) == target );
}
/**
 * Non-recursive findDescendent()
 */
public void test_findDescendent_2() throws Exception {
  FEV fev = new FEVImpl();
  FEV[] children = new FEV[] { fev.newChild("child 1"),
                               fev.newChild("child 2") };
  FEV target = children[0].newChild("child 11");
  target.newChild("child 111");
  children[0].newChild("child 12");

  assertTrue( ! (fev.findDescendent("child 111", false) == target) );
}
/**
 * Recursive findDescendent(), child found at first level.
 */
public void test_findDescendent_3() throws Exception {
  FEV fev = new FEVImpl();
  FEV[] children = new FEV[] { fev.newChild("child 1"),
                               fev.newChild("child 2") };
  FEV target = children[0].newChild("child 11");
  target.newChild("child 111");
  children[0].newChild("child 12");

  assertTrue( fev.findDescendent("CHild 2", true) == children[1] );
}
/**
 * Recursive hasDescendent()
 */
public void test_hasDescendent_1() throws Exception {
  FEVImpl fev = new FEVImpl();
  FEV[] children = new FEV[] { fev.newChild("child 1"),
                               fev.newChild("child 2") };
  FEV target = children[0].newChild("child 11");
  target.newChild("child 111");
  children[0].newChild("child 12");

  assertTrue( fev.hasDescendent("CHild 111", true) );
}
/**
 * Non-recursive hasDescendent()
 */
public void test_hasDescendent_2() throws Exception {
  FEVImpl fev = new FEVImpl();
  FEV[] children = new FEV[] { fev.newChild("child 1"),
                               fev.newChild("child 2") };
  FEV target = children[0].newChild("child 11");
  target.newChild("child 111");
  children[0].newChild("child 12");

  assertTrue( ! fev.hasDescendent("child 111", false) );
}

/**
 * Test newChild with duplicate name.
 */
public void test_newChild_duplicate_1() {
  try {
    FEV fev = new FEVImpl();
    FEV[] children = new FEV[] { fev.newChild("child 1"),
                                 fev.newChild("child 2") };
    FEV target = children[0].newChild("child 11");
    FEV c = target.newChild("child 111");
    fev.newChild(c.getFEVName());
    fail(DuplicateKeyException.class.getName() + " expected, but not thrown" );
  }
  catch (Exception exc) {
    assertEquals(  DuplicateKeyException.class, exc.getClass() );
  }
}

/**
 * Recursive findDescendentCCR()
 */
public void test_findDescendentCCR_1() throws Exception {
  String systemName = "System to be found";
  String ccrName = "CCR Name to be found";
  String urlString = "http://findme.com/found.xsd";
  FEV fev = new FEVImpl();
  FEV[] children = new FEV[] { fev.newChild("child 1"),
                               fev.newChild("child 2") };
  FCR[] fcrArray = new FCR[] { children[0].createFCR("FCR 0"),
                               children[1].createFCR("FCR 0") };
  CCR[] ccrArray
      = new CCR[] { fcrArray[0].newCCR( "System0", "Sys0 CCR0",
                                        new URL("http://a.com/a.xsd") ),
                    fcrArray[1].newCCR( "System1", "Sys1 CCR1",
                                        new URL("http://b.com/b.xsd") )
                  };

  CCR target = fcrArray[1].newCCR( systemName, ccrName, new URL(urlString) );
  assertTrue( fev.findDescendentCCR(systemName, new URL(urlString)) == target );
}
/**
 * Helper method to create a FEV tree for testing, all names will
 * begin with the specified string prefix.
 * Returns the CCR that is to be found.
 */
public static CCR createFEVTreeCCR(
                                FEV fev,
                                String prefix,
                                String systemName,
                                String ccrName,
                                String urlString
                                ) throws Exception {
  FEV[] children = new FEV[] { fev.newChild("child 1"),
                               fev.newChild("child 2") };
  FEV[] grandChildren = new FEV[] { children[0].newChild("child 0-0"),
                                    children[0].newChild("child 0-1"),
                                    children[1].newChild("child 1-0"),
                                    children[1].newChild("child 1-1"),
                                    children[1].newChild("child 1-2")
                                     };
  FCR[] fcrArray = new FCR[] { children[0].createFCR("FCR 0"),
                               children[1].createFCR("FCR 0"),
                               grandChildren[0].createFCR("FCR 0"),
                               grandChildren[1].createFCR("FCR 0"),
                               grandChildren[2].createFCR("FCR 0"),
                               grandChildren[3].createFCR("FCR 0"),
                               grandChildren[4].createFCR("FCR 0")
                             };
  CCR[] ccrArray = new CCR[fcrArray.length];
  for (int i=0; i<ccrArray.length; i++) {
    String indexStr = String.valueOf(i);
    ccrArray[i] = fcrArray[i].newCCR( "System" + indexStr,
                                "Sys CCR " + indexStr,
                            new URL("http://a.com/" + indexStr + ".xsd"));
  }
  CCR target = fcrArray[3].newCCR( systemName, ccrName, new URL(urlString) );
  return target;
}
/**
 * Recursive findDescendentCCR(), target in grandchild
 */
public void test_findDescendentCCR_2() throws Exception {
  String systemName = "System to be found";
  String ccrName = "CCR Name to be found";
  String urlString = "http://findme.com/found.xsd";
  FEV fev = new FEVImpl();
  CCR target = createFEVTreeCCR(fev, "abc", systemName, ccrName, urlString );
  assertTrue( fev.findDescendentCCR(systemName, new URL(urlString)) == target );
}
/**
 * Recursive findDescendentCCR(), target in grandchild
 */
public void test_findDescendentCCR_3() throws Exception {
  String systemName = "System to be found";
  String ccrName = "CCR Name to be found";
  String urlString = "http://findme.com/found.xsd";
  FEV fev = new FEVImpl();
  FCR fcr = fev.createFCR("FCR");
  CCR target = fcr.newCCR(systemName, ccrName, new URL(urlString));
  assertTrue( fev.findDescendentCCR(systemName, new URL(urlString)) == null );
}
/**
 * Recursive findDescendentCCR(), no match, null expected.
 */
public void test_findDescendentCCR_4() throws Exception {
  String systemName = "System to be found";
  String ccrName = "CCR Name to be found";
  String urlString = "http://findme.com/found.xsd";
  FEV fev = new FEVImpl();
  FEV[] children = new FEV[] { fev.newChild("child 1"),
                            fev.newChild("child 2") };
  FEV[] grandChildren = new FEV[] { children[0].newChild("child 0-0"),
                                    children[0].newChild("child 0-1"),
                                    children[1].newChild("child 1-0"),
                                    children[1].newChild("child 1-1"),
                                    children[1].newChild("child 1-2")
                                     };
  FCR[] fcrArray = new FCR[] { children[0].createFCR("FCR 0"),
                               children[1].createFCR("FCR 0"),
                               grandChildren[0].createFCR("FCR 0"),
                               grandChildren[1].createFCR("FCR 0"),
                               grandChildren[2].createFCR("FCR 0"),
                               grandChildren[3].createFCR("FCR 0"),
                               grandChildren[4].createFCR("FCR 0"),
                             };
  CCR[] ccrArray = new CCR[fcrArray.length];
  for (int i=0; i<ccrArray.length; i++) {
    String indexStr = String.valueOf(i);
    ccrArray[i] = fcrArray[i].newCCR( "System" + indexStr,
                                   "Sys CCR " + indexStr,
                               new URL("http://a.com/" + indexStr + ".xsd"));
  }
  CCR target = fcrArray[3].newCCR( systemName, ccrName, new URL(urlString) );
  assertTrue( fev.findDescendentCCR(systemName + "a", new URL(urlString)) == null );
}

/**
 * Recursive findCCR(), target in its own CCRs.
 */
public void test_findCCR_1() throws Exception {
  String systemName = "System to be found";
  String ccrName = "CCR Name to be found";
  String urlString = "http://findme.com/found.xsd";
  FEV fev = new FEVImpl();
  FCR fcr = fev.createFCR("FCR");
  CCR target = fcr.newCCR(systemName, ccrName, new URL(urlString));
  assertTrue( fev.findCCR(systemName, new URL(urlString)) == target );
}
/**
 * Recursive findCCR(), target in its grandchildren's CCRs.
 */
public void test_findCCR_2() throws Exception {
  String systemName = "System to be found";
  String ccrName = "CCR Name to be found";
```

```
    String urlString = "http://findme.com/found.xsd";
    FEV rootFEV = new FEVImpl();
    FCR fcr = rootFEV.createFCR("FCR");
    // level 1
    String suffix = "1";
    FEV fev = rootFEV.newChild("FEV" + suffix);
      fcr = fev.createFCR( "FCR" + suffix );
      fcr.newCCR( "System" + suffix, "CCR" + suffix,
                  new URL("http://a.com/0.xsd" + suffix) );
    // level 2
    suffix = "2";
    fev = fev.newChild("FEV" + suffix);
      fcr = fev.createFCR( "FCR" + suffix );
      fcr.newCCR( "System" + suffix, "CCR" + suffix,
                  new URL("http://a.com/0.xsd" + suffix) );

    CCR target = fcr.newCCR(systemName, ccrName, new URL(urlString));
    assertTrue( rootFEV.findCCR(systemName, new URL(urlString)) == target );
  }

} //public class ZzzTest_FEVImpl extends TestCase
```

20. ZzzTest_FIOMImpl.java

```
package mil.navy.nps.cs.oomi.impl;
import java.util.*;
import java.net.URL;

import junit.framework.TestCase;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

public class ZzzTest_FIOMImpl extends TestCase
{

    public ZzzTest_FIOMImpl(String Name_)
    {
        super(Name_);
    } //public ZzzTest_FIOMImpl(String Name_)


    FE[] feArray;
    FIOMImpl fiomImpl;
```

```
  FIOM fiom;
  protected void setUp()
  {
    feArray = new FE[] { new FEImpl(null, null, "FE 0"),
                         new FEImpl(null, null, "FE 1") };
    fiomImpl = new FIOMImpl("A FIOM");
    fiomImpl.setFE(feArray);
    fiom = fiomImpl;
  } //protected void setUp()

  protected void tearDown()
  {
  } //protected void tearDown()

  public static void main(String[] args)
  {
      String[] testCaseName = {ZzzTest_FIOMImpl.class.getName()};
      junit.swingui.TestRunner.main(testCaseName);
  } //public static void main(String[] args)

public void test_1() {
  String name = "my name";
  FIOM fiom = new FIOMImpl("my name");
  assertEquals( name, fiom.getFIOMName() );
}

public void test_equals_true() {
  FIOM fiom1 = new FIOMImpl("my name");
  FIOM fiom2 = new FIOMImpl("my name");
  assertEquals( fiom1, fiom2 );
}
public void test_equals_false_1() {
  FIOM fiom1 = new FIOMImpl("my name1");
  FIOM fiom2 = new FIOMImpl("my name");
  assertTrue( ! fiom1.equals(fiom2) );
}
public void test_equals_false_2() throws Exception {
  FIOM fiom1 = new FIOMImpl("my name");
  fiom1.newFE( "fe1" );
  FIOM fiom2 = new FIOMImpl(fiom1.getFIOMName());
  fiom2.newFE( "fe2" );
  assertTrue( ! fiom1.equals(fiom2) );
}
```

```
public void test_equals_null() {
  FIOM fiom1 = new FIOMImpl("my name1");
  assertTrue( ! fiom1.equals(null) );
}
public void test_feCount_1() {
  FIOM fiom = new FIOMImpl("A FIOM");
  assertEquals( 0, fiom.feCount() );
}
public void test_getFE_setFE_1() {
  FE[] feArray = new FE[] { new FEImpl(null, null, "FE 1"),
                            new FEImpl(null, null, "FE 2") };
  FIOMImpl fiom = new FIOMImpl("A FIOM");
  fiom.setFE(feArray);
  FE[] testee = fiom.getFE();
  assertTrue( Arrays.equals( feArray, testee ) &&
              testee[0].getFIOM()==fiom &&
              testee[1].getFIOM()==fiom );
}
/**
 * Tests calling setFE() twice.
 */
public void test_getFE_setFE_2() {
  FE[] feArray = new FE[] { new FEImpl(null, null, "FE 1"),
                            new FEImpl(null, null, "FE 2") };
  FIOMImpl fiom = new FIOMImpl("A FIOM");
  fiom.setFE(feArray);
  fiom.setFE(feArray);
  FE[] testee = fiom.getFE();
  assertTrue( Arrays.equals( feArray, testee ) &&
              testee[0].getFIOM()==fiom &&
              testee[1].getFIOM()==fiom );
}
public void test_findFE_1() {
  FE[] feArray = new FE[] { new FEImpl(null, null, "FE 0"),
                            new FEImpl(null, null, "FE 1") };
  FIOMImpl fiom = new FIOMImpl("A FIOM");
  fiom.setFE(feArray);
  assertTrue( fiom.findFE("fe 0") == feArray[0] &&
              fiom.findFE("fe a") == null );
}
public void test_feExists_true() {
  assertEquals( true, fiomImpl.feExists("fe 1") );
}
```

```
public void test_feExists_false() {
  assertEquals( false, fiomImpl.feExists("a fe 1") );
}
public void test_newFE_1 () throws Exception {
  int c = fiom.feCount();
  FE fe = fiom.newFE( "a new FE" );
  assertTrue( fe == fiom.findFE("a new FE") &&
              fe.getFIOM() == fiom &&
              fiom.feCount() == c+1 );
}
public void test_newFE_duplicate_1 () throws Exception {
  try {
    FE fe = fiom.newFE( "a new FE" );
    fiom.newFE( "a new FE" );
    fail(DuplicateKeyException.class.getName() + " expected but not thrown.");
  }
  catch (Exception exc) {
    assertEquals( DuplicateKeyException.class, exc.getClass() );
  }
}

public void test_findCCR_1() throws Exception {
  String systemName = "System to be found";
  String ccrName = "CCR Name to be found";
  String urlString = "http://findme.com/found.xsd";

  FE[] feArray = new FE[] { new FEImpl(null, null, "FE 0"),
                            new FEImpl(null, null, "FE 1") };
  FIOMImpl fiom = new FIOMImpl("A FIOM");
  fiom.setFE(feArray);

  FE fe;
  FCR fcr;

  // level 1
  fe = feArray[0];
  String suffix = "0";
  fe = fe.newChild( "Child" + suffix );
    fcr = fe.createRootFEV( "FEV" + suffix ).createFCR( "FCR" + suffix );
    fcr.newCCR( "System" + suffix, "CCR" + suffix,
                new URL("http://a.com/0.xsd" + suffix) );
  // level 1
  fe = feArray[1];
  suffix = "0";
  fe = fe.newChild( "Child" + suffix );
    fcr = fe.createRootFEV( "FEV" + suffix ).createFCR( "FCR" + suffix );
    fcr.newCCR( "System" + suffix, "CCR" + suffix,
                new URL("http://a.com/0.xsd" + suffix) );

  CCR target = fcr.newCCR(systemName, ccrName, new URL(urlString));

  assertTrue( fiom.findCCR(systemName, new URL(urlString)) == target );
}

/**
 * Compares the two arrays, equals if and only if
 *  obj1.length==obj2.length and
 *  for each obj1[i], there exists exactly one obj2[j] such that
 *    obj1[i].equals(obj2[i])
 *  and
 *  for each obj2[i], there exists exactly one obj1[j] such that
 *    obj2[i].equals(obj1[i])
public boolean equalsUnordered(Object[] obj1, Object[] obj2) {
  if (obj1.length!=obj2.length)
    return false;
  ArrayList list1 = new ArrayList( Arrays.asList(obj1) );
  Iterator iter
  for (int i=0; i<obj2.length; i++) {

  }
}
*/

/**
 * Test findTranslation.
 */
public void test_findTranslation() {
  String[] trans = new String[] { "a.fcr1,a.ccr1=trans1",
                                  "a.fcr2,a.ccr2=trans2" };
  FIOMImpl fiom = new FIOMImpl();
  fiom.setTranslations( trans );
  String[] testee = fiom.getTranslations();
  assertTrue( fiom.findTranslation("a.fcr1", "a.ccr1").equals("trans1") );
}
/**
 * Test get/set Translations.
```

```
   */
  public void test_getsetTranslations() {
    String[] trans = new String[] { "a.fcr1,a.ccr1=trans1",
                                    "a.fcr2,a.ccr2=trans2" };
    FIOMImpl fiom = new FIOMImpl();
    fiom.setTranslations( trans );
    String[] testee = fiom.getTranslations();
    Arrays.sort(testee);
    Arrays.sort(trans);
    assertTrue( Arrays.equals( trans, testee ) );
  }

  /**
   * Test findTranslation.
   */
  //public void test_findTranslation

  public void test_registerCCR_1() throws Exception {
    FIOM fiom1 = new FIOMImpl("my name");
    FE fe = fiom1.newFE("FE");
    FEV fev = fe.createRootFEV("ROOT");
    FCR fcr = fev.createFCR("FCR");
    CCR unregistered1 = fiom1.newCCR( "A", "CCR A1", new URL("http://a.b.c/A1") );
    CCR unregistered2 = fiom1.newCCR( "A", "CCR A2", new URL("http://a.b.c/A2") );
    int countBeforeReg = fiom1.unregisteredCCRCount();
    fiom1.registerCCR(fcr, unregistered1);
    assertTrue( fiom1.unregisteredCCRCount()==1
                && countBeforeReg==2
                && fcr.getCCR()[0]==unregistered1
              );
  }
} //public class ZzzTest_FIOMImpl extends TestCase
```

21. ZzzTest_OOMIDatabaseImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import java.util.*;
import java.io.*;
import java.net.*;
import junit.framework.TestCase;
import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

public class ZzzTest_OOMIDatabaseImpl extends TestCase
{

  FIOMFactory _factory = FactoryImpl.newFIOMFactory();

    public ZzzTest_OOMIDatabaseImpl(String Name_)
    {
        super(Name_);
    } //public ZzzTest_OOMIDatabaseImpl(String Name_)

    OOMIDatabase oomiDb = null;
    String[] fiomNameArray = null;

    protected void setUp() throws Exception
    {
      oomiDb = OOMIDatabaseImpl.loadFromFile("");
      fiomNameArray = new String[] { "1", "2" };
    } //protected void setUp()

    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_OOMIDatabaseImpl.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)


  public void test_saveToFile_1() throws Exception {
    File file = File.createTempFile("OOMIDatabaseImpl_test_saveToFile_1", ".xml");
    file.deleteOnExit();
    oomiDb.newFIOM("New FiOm");
    ((OOMIDatabaseImpl)oomiDb).saveToFile(file.getAbsolutePath());
    OOMIDatabase db = OOMIDatabaseImpl.loadFromFile(file.getAbsolutePath());
    assertEquals( oomiDb, db );
  }
  public void test_saveToFile_2_FIOMtoFE() throws Exception {
    File file = File.createTempFile("OOMIDatabaseImpl_test_saveToFile_2_FIOMtoFE", ".xml");
    file.deleteOnExit();
    oomiDb.newFIOM("New FiOm");
    oomiDb.newFIOM("New FiOm A");
      FIOM fiom = oomiDb.newFIOM(" FIOM   1 "); // with trailing, beginning and double space
      FE fe = fiom.newFE( "FIOM 1 FE 1" );
      fiom.newFE( "FIOM 1 FE 2" );
      ((OOMIDatabaseImpl)oomiDb).saveToFile(file.getAbsolutePath());
    OOMIDatabase db = OOMIDatabaseImpl.loadFromFile(file.getAbsolutePath());
      FIOM[] fiomsA = oomiDb.getFIOM();
      FIOM[] fiomsB = db.getFIOM();

      FIOM fiomB = db.findFIOM(" FIOM   1 ");
      FE feB = fiomB.findFE("FIOM 1 FE 1");
    assertTrue( fiomsA.length == fiomsB.length && feB.getFIOM() == fiomB  );
  }
  public void test_saveToFile_2_FIOM_UnregisteredCCRs() throws Exception {
    File file = File.createTempFile("test_saveToFile_2_FIOM_UnregisteredCCRs", ".xml");
    file.deleteOnExit();
    oomiDb.newFIOM("New FiOm");
    oomiDb.newFIOM("New FiOm A");
      FIOM fiom = oomiDb.newFIOM(" FIOM   1 "); // with trailing, beginning and double space
        CCR ccr1A = fiom.newCCR( "Sys A", "CCR 1A", new URL( "http://a.b.c/1A") );
        CCR ccr1B = fiom.newCCR( "Sys A", "CCR 1B", new URL( "http://a.b.c/1B") );
      FE fe = fiom.newFE( "FIOM 1 FE 1" );
      fiom.newFE( "FIOM 1 FE 2" );
      ((OOMIDatabaseImpl)oomiDb).saveToFile(file.getAbsolutePath());
    OOMIDatabase db = OOMIDatabaseImpl.loadFromFile(file.getAbsolutePath());

      FIOM loadedFiom1 = db.findFIOM(" FIOM   1 ");
      CCR loadedCCR1A = loadedFiom1.findUnregisteredCCR(ccr1A.getSystemName(),
                                                  ccr1A.getCCRName() );
      CCR loadedCCR1B = loadedFiom1.findUnregisteredCCR(ccr1B.getSystemName(),
                                                  ccr1B.getCCRName() );
    assertTrue(  loadedCCR1A.getSystemName().equals(ccr1A.getSystemName())
              && loadedCCR1A.getCCRName().equals(ccr1A.getCCRName())
              && loadedCCR1A.getXMLNameSpaceURI().equals(ccr1A.getXMLNameSpaceURI())
              && loadedCCR1B.getSystemName().equals(ccr1B.getSystemName())
              && loadedCCR1B.getCCRName().equals(ccr1B.getCCRName())
              && loadedCCR1B.getXMLNameSpaceURI().equals(ccr1B.getXMLNameSpaceURI())
               );
  }
  public void test_saveToFile_FIOMtoFEV_1() throws Exception {
    File file = File.createTempFile("OOMIDatabaseImpl_test_saveToFile_FIOMtoFEV_1", ".xml");
    file.deleteOnExit();
    oomiDb.newFIOM("New FiOm");
    FIOM fiom = oomiDb.newFIOM("FIOM 1");
      FE fe = fiom.newFE( "FIOM 1 FE 1" );
      FEV fev = fe.createRootFEV( "FIOM 1 FE 1 FEV 1" );
    ((OOMIDatabaseImpl)oomiDb).saveToFile(file.getAbsolutePath());
    OOMIDatabase db = OOMIDatabaseImpl.loadFromFile(file.getAbsolutePath());
    FIOM[] fiomsA = oomiDb.getFIOM();
    FIOM[] fiomsB = db.getFIOM();

    FIOM fiomB = db.findFIOM("FIOM 1");
    FE feB = fiomB.findFE("FIOM 1 FE 1");
    FEV fevB = feB.findFEV("FIOM 1 FE 1 FEV 1");

    assertTrue( fevB.getFE()==feB );
  }
  public void test_saveToFile_FIOMtoFCR_1() throws Exception {
    File file = File.createTempFile("OOMIDatabaseImpl_test_saveToFile_1", ".xml");
    file.deleteOnExit();
    oomiDb.newFIOM("New FiOm");
    FIOM fiom = oomiDb.newFIOM("FIOM 1");
      FE fe = fiom.newFE( "FIOM 1 FE 1" );
      FEV fev = fe.createRootFEV( "FIOM 1 FE 1 FEV 1" );
      FCR fcr = fev.createFCR("FIOM 1 FE 1 FEV 1 fcr 1");
    ((OOMIDatabaseImpl)oomiDb).saveToFile(file.getAbsolutePath());
    OOMIDatabase db = OOMIDatabaseImpl.loadFromFile(file.getAbsolutePath());
    FIOM[] fiomsA = oomiDb.getFIOM();
    FIOM[] fiomsB = db.getFIOM();

    FIOM fiomB = db.findFIOM("FIOM 1");
    FE feB = fiomB.findFE("FIOM 1 FE 1");
    FEV fevB = feB.findFEV("FIOM 1 FE 1 FEV 1");

    assertEquals( "FIOM 1 FE 1 FEV 1 fcr 1", fevB.getFCR().getFCRName() );
    //                fcr.getFEV()==fevB );
  }
  /**
   * Checks if the FEV property of an FCR
   * is properly set after loading from file.
   */
  public void test_saveToFile_FIOMtoFCR_2() throws Exception {
    File file = File.createTempFile("OOMIDatabaseImpl_test_saveToFile_1", ".xml");
    file.deleteOnExit();
    oomiDb.newFIOM("New FiOm");
    FIOM fiom = oomiDb.newFIOM("FIOM 1");
      FE fe = fiom.newFE( "FIOM 1 FE 1" );
      FEV fev = fe.createRootFEV( "FIOM 1 FE 1 FEV 1" );
      FCR fcr = fev.createFCR("FIOM 1 FE 1 FEV 1 fcr 1");
    ((OOMIDatabaseImpl)oomiDb).saveToFile(file.getAbsolutePath());
    OOMIDatabase db = OOMIDatabaseImpl.loadFromFile(file.getAbsolutePath());
    FIOM[] fiomsA = oomiDb.getFIOM();
    FIOM[] fiomsB = db.getFIOM();

    FIOM fiomB = db.findFIOM("FIOM 1");
    FE feB = fiomB.findFE("FIOM 1 FE 1");
    FEV fevB = feB.findFEV("FIOM 1 FE 1 FEV 1");

    assertTrue( fevB.getFCR().getFEV()==fevB );
  }
  /**
   * Checks if CCR is properly restored from storage.
   */
  public void test_saveToFile_FIOMtoCCR_1() throws Exception {
    File file = File.createTempFile("OOMIDatabaseImpl_test_saveToFile_FIOMtoCCR_1-", ".xml");
    file.deleteOnExit();
    oomiDb.newFIOM("New FiOm");
    FIOM fiom = oomiDb.newFIOM("FIOM 1");
      FE fe = fiom.newFE( "FIOM 1 FE 1" );
      FEV fev = fe.createRootFEV( "FIOM 1 FE 1 FEV 1" );
      FCR fcr = fev.createFCR("FIOM 1 FE 1 FEV 1 fcr 1");
      CCR ccr = fcr.newCCR("System", "FIOM 1 FE 1 FEV 1 fcr 1 ccr 1", null );
    ((OOMIDatabaseImpl)oomiDb).saveToFile(file.getAbsolutePath());
    OOMIDatabase db = OOMIDatabaseImpl.loadFromFile(file.getAbsolutePath());
    FIOM[] fiomsA = oomiDb.getFIOM();
    FIOM[] fiomsB = db.getFIOM();

    FIOM fiomB = db.findFIOM("FIOM 1");
    FE feB = fiomB.findFE("FIOM 1 FE 1");
    FEV fevB = feB.findFEV("FIOM 1 FE 1 FEV 1");
    FCR fcrB = fevB.getFCR();
    CCR ccrB = (fcrB.getCCR())[0];
    assertEquals( "FIOM 1 FE 1 FEV 1 fcr 1 ccr 1", ccrB.getCCRName() );
  }
  /**
   * Checks if CCR is properly restored from storage.
   */
  public void test_saveToFile_FIOMtoCCR_2() throws Exception {
    File file = File.createTempFile("OOMIDatabaseImpl_test_saveToFile_FIOMtoCCR_2-", ".xml");
    file.deleteOnExit();
    oomiDb.newFIOM("New FiOm");
    FIOM fiom = oomiDb.newFIOM("FIOM 1");
      FE fe = fiom.newFE( "FIOM 1 FE 1" );
      FEV fev = fe.createRootFEV( "FIOM 1 FE 1 FEV 1" );
      FCR fcr = fev.createFCR("FIOM 1 FE 1 FEV 1 fcr 1");
      CCR ccr = fcr.newCCR("System", "FIOM 1 FE 1 FEV 1 fcr 1 ccr 1", null );
      fcr.newCCR("System", "FIOM 1 FE 1 FEV 1 fcr 1 ccr 2", null );
    ((OOMIDatabaseImpl)oomiDb).saveToFile(file.getAbsolutePath());

    OOMIDatabase db = OOMIDatabaseImpl.loadFromFile(file.getAbsolutePath());

    FIOM[] fiomsB = db.getFIOM();

    FIOM fiomB = db.findFIOM("FIOM 1");
    FE feB = fiomB.findFE("FIOM 1 FE 1");
    FEV fevB = feB.findFEV("FIOM 1 FE 1 FEV 1");
    FCR fcrB = fevB.getFCR();
    CCR ccrB = (fcrB.getCCR())[0];

    assertTrue( ccrB.getFCR()==fcrB );
  }

  /**
   * Checks if FCRSchema is properly restored from storage.
   * Tests if the Parent property of the attributes are properly restored.
   * Other properties should be properly restored, since they are "primitive"
   * types.
   */
  public void test_saveToFile_FIOMtoFCRSchema_1() throws Exception {
    File file = File.createTempFile("OOMIDatabaseImpl_test_saveToFile_1", ".xml");
    file.deleteOnExit();
    oomiDb.newFIOM("New FiOm");
    FIOM fiom = oomiDb.newFIOM("FIOM 1");
      FE fe = fiom.newFE( "FIOM 1 FE 1" );
      FEV fev = fe.createRootFEV( "FIOM 1 FE 1 FEV 1" );
      FCR fcr = fev.createFCR("FIOM 1 FE 1 FEV 1 fcr 1");
        FCRSchema fcrSchema = fcr.getFCRSchema();
          fcrSchema.setType(_factory.makeTypeName("The.root.type") );
          Attribute      fcrAttr_1      =      _factory.makeAttribute(null,      "fcrAttr_1",
_factory.makeTypeName("anyType"));
          fcrSchema.addAttribute( fcrAttr_1 );
          Attribute      fcrAttr_2      =      _factory.makeAttribute(null,      "fcrAttr_2",
_factory.makeTypeName("anyType"));
          fcrSchema.addAttribute( fcrAttr_2 );
            fcrAttr_2.newChild("a", _factory.makeTypeName("t1"));
            fcrAttr_2.newChild("b", _factory.makeTypeName("t2"));
      CCR ccr = fcr.newCCR("System", "FIOM 1 FE 1 FEV 1 fcr 1 ccr 1", null );
    ((OOMIDatabaseImpl)oomiDb).saveToFile(file.getAbsolutePath());
    OOMIDatabase db = OOMIDatabaseImpl.loadFromFile(file.getAbsolutePath());
    FIOM[] fiomsA = oomiDb.getFIOM();
    FIOM[] fiomsB = db.getFIOM();

    FIOM fiomB = db.findFIOM("FIOM 1");
    FE feB = fiomB.findFE("FIOM 1 FE 1");
    FEV fevB = feB.findFEV("FIOM 1 FE 1 FEV 1");
    FCR fcrB = fevB.getFCR();
    CCR ccrB = (fcrB.getCCR())[0];

    FCRSchema loadedFCRSchema = fcrB.getFCRSchema();
      //Attribute loadedFcrRootAttr = loadedFCRSchema.getRootAttribute();
      Attribute[] loadedFcrAttr = loadedFCRSchema.getAttribute();
      Attribute[] secondLevel = loadedFcrAttr[1].getChild();
    assertTrue( loadedFCRSchema.getType().equals(_factory.makeTypeName("The.root.type")) &&
                loadedFcrAttr[0].equals(fcrAttr_1) &&
                loadedFcrAttr[1].equals(fcrAttr_2) );
  }
  /**
   * Checks if CCRSchema is properly restored from storage.
   * Tests if the Parent property of the attributes are properly restored.
   * Other properties should be properly restored, since they are "primitive"
   * types.
   */
  public void test_saveToFile_FIOMtoCCRSchema_1() throws Exception {
    File file = File.createTempFile("OOMIDatabaseImpl_test_saveToFile_1", ".xml");
    file.deleteOnExit();
    oomiDb.newFIOM("New FiOm");
    FIOM fiom = oomiDb.newFIOM("FIOM 1");
      FE fe = fiom.newFE( "FIOM 1 FE 1" );
      FEV fev = fe.createRootFEV( "FIOM 1 FE 1 FEV 1" );
      FCR fcr = fev.createFCR("FIOM 1 FE 1 FEV 1 fcr 1");
        FCRSchema fcrSchema = fcr.getFCRSchema();
          fcrSchema.setType(_factory.makeTypeName("The.root.type"));
          Attribute      fcrAttr_1      =      _factory.makeAttribute(null,      "fcrAttr_1",
_factory.makeTypeName("anyType"));
          fcrSchema.addAttribute( fcrAttr_1 );
          Attribute      fcrAttr_2      =      _factory.makeAttribute(null,      "fcrAttr_2",
_factory.makeTypeName("anyType"));
          fcrSchema.addAttribute( fcrAttr_2 );
            fcrAttr_2.newChild("a", _factory.makeTypeName("t1"));
            fcrAttr_2.newChild("b", _factory.makeTypeName("t2"));
      CCR ccr = fcr.newCCR("System", "FIOM 1 FE 1 FEV 1 fcr 1 ccr 1", null );
        CCRSchema ccrSchema = ccr.getCCRSchema();
          ccrSchema.setType(_factory.makeTypeName("The.root.type"));
          Attribute      ccrAttr_1      =      _factory.makeAttribute(null,      "ccrAttr_1",
_factory.makeTypeName("anyType"));
            ccrSchema.addAttribute(ccrAttr_1);
          Attribute      ccrAttr_2      =      _factory.makeAttribute(null,      "ccrAttr_2",
_factory.makeTypeName("anyType"));
            ccrSchema.addAttribute(ccrAttr_2);
            ccrAttr_2.newChild("a", _factory.makeTypeName("t1"));
            ccrAttr_2.newChild("b", _factory.makeTypeName("t2"));
    ((OOMIDatabaseImpl)oomiDb).saveToFile(file.getAbsolutePath());
    OOMIDatabase db = OOMIDatabaseImpl.loadFromFile(file.getAbsolutePath());
    FIOM[] fiomsA = oomiDb.getFIOM();
    FIOM[] fiomsB = db.getFIOM();

    FIOM fiomB = db.findFIOM("FIOM 1");
    FE feB = fiomB.findFE("FIOM 1 FE 1");
    FEV fevB = feB.findFEV("FIOM 1 FE 1 FEV 1");
    FCR fcrB = fevB.getFCR();
    CCR ccrB = (fcrB.getCCR())[0];

    CCRSchema loadedCCRSchema = ccrB.getCCRSchema();
      Attribute[] loadedCcrAttr = loadedCCRSchema.getAttribute();
      Attribute[] secondLevel = loadedCcrAttr[1].getChild();
    assertTrue( loadedCCRSchema.getType().equals(_factory.makeTypeName("The.root.type")) &&
                loadedCcrAttr[0].equals(ccrAttr_1) &&
                loadedCcrAttr[1].equals(ccrAttr_2) );
  }

  public void test_1() {
    assertEquals(oomiDb.findFIOM("no such fiom"), null);
  }
  /**
   * Tests the empty case.
   */
  public void test_empty_count() {
    assertEquals(0, oomiDb.fiomCount());
  }
```

```
public void test_newFIOM_1() throws Exception {
  int c = oomiDb.fiomCount();
  oomiDb.newFIOM("New FiOm");
  assertEquals(1, oomiDb.fiomCount());
}
public void test_fiomExists_true() throws Exception {
  oomiDb.newFIOM("New FiOm");
  assertTrue( oomiDb.fiomExists("new fioM") &&
              !oomiDb.fiomExists("a") &&
              !oomiDb.fiomExists(""));
}
public void test_newFIOM_duplicate_1() throws Exception {
  oomiDb.newFIOM("New FiOm");
  int c = oomiDb.fiomCount();
  try {
    oomiDb.newFIOM("New FiOm");
    fail("Exception expected, but was not raised");
  }
  catch (Exception e) {
    assertEquals(c, oomiDb.fiomCount());
  }
}
/**
 * Test the case-insensitive find of existing FIOM.
 */
public void test_findFIOM_1() throws Exception {
  int c = oomiDb.fiomCount();
  oomiDb.newFIOM("New FiOm 1");
  oomiDb.newFIOM("New FiOm");
  oomiDb.newFIOM("New FiOm 2");
  assertTrue(oomiDb.findFIOM("new fiom") != null);
}
public void test_getFIOM_empty_array() {
  assertEquals( 0, oomiDb.getFIOM().length );
}
public void test_getFIOM_nonempty_array() throws OOMIException {
  FIOM[] fioms = new FIOMImpl[fiomNameArray.length];
  for (int i=0; i<fiomNameArray.length; i++) {
    fioms[i]=oomiDb.newFIOM(fiomNameArray[i]);
  }
  assertTrue( Arrays.equals(fioms, oomiDb.getFIOM())  );
}
public void test_equals_null() {
    assertTrue( ! oomiDb.equals(null) );
  }
  public static void p1(String m) {
    System.out.println(m);
  }
} //public class ZzzTest_OOMIDatabaseImpl extends TestCase
```

22. ZzzTest_SchemaImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import junit.framework.TestCase;
import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;

public class ZzzTest_SchemaImpl extends TestCase
{

    public ZzzTest_SchemaImpl(String Name_)
    {
        super(Name_);
    } //public ZzzTest_SchemaImpl(String Name_)

    protected void setUp()
    {
    } //protected void setUp()

    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_SchemaImpl.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)

  FIOMFactory _factory = FactoryImpl.factory();

  public void test_findAttribute_1() throws Exception {
    Schema s = new SchemaImpl();
    Attribute attr = s.addAttribute( "The name", _factory.makeTypeName("The Type"));
    Attribute attr_1 = attr.newChild( "name1", _factory.makeTypeName("type"));
    Attribute attr_1_1 = attr_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute attr_2 = attr.newChild( "name2", _factory.makeTypeName("type"));
    Attribute attr_3 = attr.newChild( "The name", _factory.makeTypeName("type"));
    Attribute attr_4 = attr.newChild( "The name 4", _factory.makeTypeName("type"));

    Attribute  a  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type"));
    Attribute a_1 = a.newChild( "name1", _factory.makeTypeName("type"));
    Attribute a_1_1 = a_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute a_2 = a.newChild( "name2", _factory.makeTypeName("type"));
    Attribute a_3 = a.newChild( "The name", _factory.makeTypeName("type"));
    Attribute a_4 = a.newChild( "The name 4", _factory.makeTypeName("type"));

    assertTrue( s.findAttribute(a_1_1).equals(attr_1_1) &&
                s.findAttribute(a_1).equals(attr_1) );
  }

  public void test_findAttribute_2() throws Exception {
    Schema s = new SchemaImpl();

    Attribute attr = s.addAttribute( "The name", _factory.makeTypeName("The Type"));
    Attribute attr_1 = attr.newChild( "name1", _factory.makeTypeName("type"));
    Attribute attr_1_1 = attr_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute attr_2 = attr.newChild( "name2", _factory.makeTypeName("type"));
    Attribute attr_3 = attr.newChild( "The name", _factory.makeTypeName("type"));
    Attribute attr_4 = attr.newChild( "The name 4", _factory.makeTypeName("type"));

    Attribute  a  =  _factory.makeAttribute(  null,  "The  name",  _factory.makeTypeName("The
Type"));
    Attribute a_1 = a.newChild( "name1", _factory.makeTypeName("type"));
    Attribute a_1_1 = a_1.newChild("name1-1",_factory.makeTypeName("type1-1"));
    Attribute a_2 = a.newChild( "name2", _factory.makeTypeName("type"));
    Attribute a_3 = a.newChild( "The name NOTFOUND", _factory.makeTypeName("type"));
    Attribute a_4 = a.newChild( "The name 4", _factory.makeTypeName("type"));

    assertTrue( s.findAttribute(a_3) == null );
  }

} //public class ZzzTest_SchemaImpl extends TestCase
```

23. ZzzTest_TypeNameImpl.java

```
package mil.navy.nps.cs.oomi.impl;

import junit.framework.TestCase;
import mil.navy.nps.cs.oomi.fiom.*;

public class ZzzTest_TypeNameImpl extends TestCase
{

    public ZzzTest_TypeNameImpl(String Name_)
    {
        super(Name_);
    } //public ZzzTest_TypeNameImpl(String Name_)

    protected void setUp()
    {
    } //protected void setUp()

    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_TypeNameImpl.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)

  public static void p(String m) {
    System.out.println(m);
  }

  FIOMFactory f = FactoryImpl.factory();
  public void test_equals_1() {
    TypeName t1 = f.makeTypeName( String.class );
    TypeName t2 = f.makeTypeName( String.class );
    assertTrue( t1.equals(t2) );

  }
  public void test_equals_2() {
    TypeName t1 = f.makeTypeName( String.class );
    TypeName t2 = f.makeTypeName( "a" );
    assertTrue( !t1.equals(t2)  );
  }
  public void test_hashCode_1() {
    TypeName t1 = f.makeTypeName( "java.lang.String" );
    TypeName t2 = f.makeTypeName( String.class.getName() );
    assertTrue( t1.hashCode() == t2.hashCode() );
  }
  public void test_hashCode_2() {
    TypeName t1 = f.makeTypeName( "java.lang.String" );
    TypeName t2 = f.makeTypeName( String.class.getName() + "1" );
    assertTrue( t1.hashCode() != t2.hashCode() );
  }
  public void test_isArray_1() {
    assertTrue( ! f.makeTypeName(String.class).isArray() );
  }
  public void test_isArray_2() {
    assertTrue( f.makeArrayTypeName(String.class, 1).isArray() );
  }
  public void test_isPrimitive_1() {
    assertTrue( !f.makeArrayTypeName(String.class, 1).isPrimitive() );
  }
  public void test_isPrimitive_2() {
    assertTrue( !f.makeArrayTypeName(Integer.TYPE, 1).isPrimitive() );
  }
  public void test_isPrimitive_3() {
    TypeName t = f.makeTypeName(Integer.TYPE);
    assertTrue( t.isPrimitive() );
  }
  public void test_isPrimitive_4() {
    assertTrue( !f.makeTypeName(Integer.class).isPrimitive() );
  }
  public void test_duplicate_1() {
    TypeName t = f.makeTypeName( "abc" );
    TypeName clone = t.copy();
    assertTrue( t!=clone && t.equals(clone) );
  }
  public void test_clone_1() {
    TypeName t = f.makeTypeName( "abc" );
    TypeName clone = (TypeName)((TypeNameImpl)t).clone();
    assertTrue( t!=clone && t.equals(clone) );
  }
} //public class ZzzTest_TypeNameImpl extends TestCase
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. TRANSLATION GENERATOR SOURCE CODE

A. PACKAGE: mil.navy.nps.cs.oomi.translator

1. AttributesTreeModel.java

```
package mil.navy.nps.cs.oomi.impl;

import junit.framework.TestCase;
import mil.navy.nps.cs.oomi.fiom.*;

public class ZzzTest_TypeNameImpl extends TestCase
{

    public ZzzTest_TypeNameImpl(String Name_)
    {
        super(Name_);
    } //public ZzzTest_TypeNameImpl(String Name_)

    protected void setUp()
    {
    } //protected void setUp()

    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_TypeNameImpl.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)

  public static void p(String m) {
    System.out.println(m);
  }

  FIOMFactory f = FactoryImpl.factory();
  public void test_equals_1() {
    TypeName t1 = f.makeTypeName( String.class );
    TypeName t2 = f.makeTypeName( String.class );
    assertTrue( t1.equals(t2) );

  }
  public void test_equals_2() {
    TypeName t1 = f.makeTypeName( String.class );
    TypeName t2 = f.makeTypeName( "a" );
    assertTrue( !t1.equals(t2)  );
  }
  public void test_hashCode_1() {
    TypeName t1 = f.makeTypeName( "java.lang.String" );
    TypeName t2 = f.makeTypeName( String.class.getName() );
    assertTrue( t1.hashCode() == t2.hashCode() );
  }
  public void test_hashCode_2() {
    TypeName t1 = f.makeTypeName( "java.lang.String" );
    TypeName t2 = f.makeTypeName( String.class.getName() + "1" );
    assertTrue( t1.hashCode() != t2.hashCode() );
  }
  public void test_isArray_1() {
    assertTrue( ! f.makeTypeName(String.class).isArray() );
  }
  public void test_isArray_2() {
    assertTrue( f.makeArrayTypeName(String.class, 1).isArray() );
  }
  public void test_isPrimitive_1() {
    assertTrue( !f.makeArrayTypeName(String.class, 1).isPrimitive() );
  }
  public void test_isPrimitive_2() {
    assertTrue( !f.makeArrayTypeName(Integer.TYPE, 1).isPrimitive() );
  }
  public void test_isPrimitive_3() {
    TypeName t = f.makeTypeName(Integer.TYPE);
    assertTrue( t.isPrimitive() );
  }
  public void test_isPrimitive_4() {
    assertTrue( !f.makeTypeName(Integer.class).isPrimitive() );
  }
  public void test_duplicate_1() {
    TypeName t = f.makeTypeName( "abc" );
    TypeName clone = t.copy();
    assertTrue( t!=clone && t.equals(clone) );
  }
```

```
  public void test_clone_1() {
    TypeName t = f.makeTypeName( "abc" );
    TypeName clone = (TypeName)((TypeNameImpl)t).clone();
    assertTrue( t!=clone && t.equals(clone) );
  }
} //public class ZzzTest_TypeNameImpl extends TestCase
```

2. AttributeTranslationMethod.java

```
package mil.navy.nps.cs.oomi.translator;

/**
 * A method definition that defines a FIOM attribute translation.
 * Contains additional information required to generate translations,
 * specifically, contains the target TranslationAttribute that this
 * TranslationMethodDefinition is meant for.
 */
public class AttributeTranslationMethod extends MethodDefinition {

  /**
   * The target attribute that this method is meant to translate to.
   */
  protected TranslationAttribute _target = null;


  public AttributeTranslationMethod() {
  }

  public AttributeTranslationMethod(
                        String modifiers,
                        String returnType,
                        String methodName,
                        TranslationAttribute targetAttribute) {
    super(modifiers, returnType, methodName);
    _target = targetAttribute;
  }
  public TranslationAttribute getTargetAttribute() {
    return _target;
  }
}
```

3. ClassDefinition.java

```
package mil.navy.nps.cs.oomi.translator;
```

```
import java.util.*;

/**
 *
 * <p>
 * An instance of this class defines a Java class.
 *
 * Currently, does not support inner classes.
 *
 * @author LSC
 * @version 1.0
 *
 */
public class ClassDefinition {

  /**
   * Modifiers for this ClassDefinition.
   */
  protected String _modifiers = "";

  /**
   * Name of the Java class defined by this object.
   */
  protected String _name = null;

  /**
   * Name of the parent Java class for the class defined,
   * defaults to the Java base class: Object.
   */
  protected String _parentName = Object.class.getName();


  /**
   * List of MethodDefinition instances.
   */
  protected List _methods = new LinkedList();

  /**
   * Custom code segment for field declarations managed by the user.
   */
  protected List _userManagedFields = new LinkedList();
```

```
/**
 * Custom code segment for method definitions managed by the user.
 */
protected List _userManagedMethods = new LinkedList();


/**
 * Constructor, if parentName is null, the parent of this ClassDefinition
 * will be the default Java Object class.
 */
public ClassDefinition(String className, String parentName) {
  setParentName(parentName);
  setName(className);
}

public void setModifiers( String modifiers ) {
  this._modifiers = modifiers;
}
public String getModifiers() {
  return this._modifiers;
}
public void setParentName( String parentName ) {
  if (parentName==null)
    this._parentName = Object.class.getName();
  else
    this._parentName = parentName;
}
public String getParentName() {
  return this._parentName;
}
public void setName( String className ) {
  this._name = className;
}
public String getName() {
  return this._name;
}
public void addMethods( List methods ) {
  this._methods.addAll(methods);
}
public void addMethod( MethodDefinition method ) {
  this._methods.add(method);
}
public void setUserManagedFields(List userFields) {
    this._userManagedFields.addAll(userFields);
  }
  public MethodDefinition[] methodsToArray() {
    int size = this._methods.size();
    if (size>0) {
      return (MethodDefinition[])
                  this._methods.toArray(new MethodDefinition[size]);
    }
    else
      return null;
  }
  public String[] userManagedFieldsToArray() {
    int size = this._userManagedFields.size();
    if (size>0)
      return (String[]) this._userManagedFields.toArray(new String[size]);
    else
      return null;
  }

}
```

4. GeneratorUI.java

```
package mil.navy.nps.cs.oomi.translator;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.io.*;
import java.util.EventObject;

import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.babel.plugin.*;

/**
 * A UI factory for the Translation generator.
 */
public class GeneratorUI  {

  protected GeneratorUI() {
  }
  /**
   * Creates a TranslationGeneratorPlugin object that handles the user
   * interface to perform mapping between FCR and CCR attributes.
   * The attribute mappings between FCR and CCR are updated to the specified
   * CCR object parameter.
   * @param container The JPanel that the user interface is to be displayed on.
   *                  If this parameter is null, this method will create a new
   *                  JPanel.
   * @param fcr The FCR for the translation skeleton to be generated.
   *            The following are required:
   *            <ol>
   *            <li> fcr.getFCRSchema() shall return a FCRSchema object
   *                  with all the attributes properly set.
   *            <li> fcr.getJavaClassName() shall return a class name of a
   *                  compiled class accessible to the TranslationGeneratorPlugin
   *                  object that is to be returned from this method invocation.
   *            <li> fcr.getFEV().getFE().getFIOM() shall return a valid reference
   *                  to the FIOM in use.
   *            </ol>
   * @param ccr The CCR for the translation skeleton to be generated.
   *            The following are required:
   *            <ol>
   *            <li> ccr.getJavaClassName() shall return a class name of a
   *                  compiled class accessible to the TranslationGeneratorPlugin
   *                  object that is to be returned from this method invocation.
   *            <li> ccr.getCCRSchema() shall return a CCRSchema object
   *                  with all the attributes properly set,
   *                  including MinOccurs and MaxOccurs properties for each
   *                  attribute.
   *            </ol>
   * @param translationPackageNameFormat The format for the package name of the
   *                               translation class to be generated.
   * @return Returns a new TranslationGeneratorPlugin object.
   */
  public static mil.navy.nps.cs.babel.plugin.TranslationGeneratorPlugin
              makePlugin(JPanel     container,     FCR     fcr,     CCR     ccr,     String
translationPackageNamePrefix)
                     throws mil.navy.nps.cs.oomi.exceptions.IntrospectionException,
                            ClassNotFoundException {
    return new Plugin( container, fcr, ccr, translationPackageNamePrefix );
  }


  static class Plugin implements mil.navy.nps.cs.babel.plugin.TranslationGeneratorPlugin {
```

```
FCR _fcr;
FIOM _fiom;
CCR _ccr;

/**
 * The package name prefix of the class to be generated.
 */
String _packageNamePrefix;

Class fcrClass;
Class ccrClass;
TranslationMap fcrToCcr;
TranslationMap ccrToFcr;
TranslationMapTableModel fcrToCcrModel;
TranslationMapTableModel ccrToFcrModel;

/**
 * Event listener list.
 */
EventListenerList _listeners = new EventListenerList();

/**
 * Array of mappings of FCR to CCR attributes,
 * this array should only be valid after the skeleton code is generated.
 */
AttributeMapping[] _fcrToCcrMapping;
/**
 * Array of mappings of CCR to FCR attributes,
 * this array should only be valid after the skeleton code is generated.
 */
AttributeMapping[] _ccrToFcrMapping;


JPanel contentPane;
JButton fcrToCcrButton = new JButton();
JButton ccrToFcrButton = new JButton();
//JScrollPane jScrollPane1 = new JScrollPane();
//JTable leftTable = new JTable();
//JScrollPane jScrollPane2 = new JScrollPane();
//JTable rightTable = new JTable();
JScrollPane fcrToCcrTableScrollPane = new JScrollPane();
JTable fcrToCcrTable = new JTable();
```

```
JScrollPane ccrToFcrTableScrollPane = new JScrollPane();
JTable ccrToFcrTable = new JTable();
JButton removeLeftToRightButton = new JButton();
JButton removeRightToLeftButton = new JButton();
JButton generateButton = new JButton();
JScrollPane fcrTreeScrollPane = new JScrollPane();
JTree fcrTree = new JTree();
JScrollPane ccrTreeScrollPane = new JScrollPane();
JTree ccrTree = new JTree();

public Plugin( Container container, FCR fcr, CCR ccr, String packageNamePrefix )
                  throws mil.navy.nps.cs.oomi.exceptions.IntrospectionException,
                         ClassNotFoundException  {
  uiInit(container);
  init(fcr, ccr, packageNamePrefix);
}

private void init(FCR fcr, CCR ccr, String packageNamePrefix)
                  throws mil.navy.nps.cs.oomi.exceptions.IntrospectionException,
                         ClassNotFoundException {
  _fcr = fcr;
  _fiom = _fcr.getFEV().getFE().getFIOM();
  _ccr = ccr;
  _packageNamePrefix = packageNamePrefix;
  fcrClass = Class.forName(fcr.getJavaClassName());
  ccrClass = Class.forName(ccr.getJavaClassName());
  fcrToCcr = new TranslationMap(fcrClass.getName(),
                                    ccrClass.getName());
  ccrToFcr = new TranslationMap(ccrClass.getName(),
                                    fcrClass.getName());
  fcrToCcrModel = new TranslationMapTableModel(fcrToCcr);
  ccrToFcrModel = new TranslationMapTableModel(ccrToFcr);

  fcrTree.setModel( new AttributesTreeModel( fcrClass ));
  ccrTree.setModel( new AttributesTreeModel( ccrClass ));



  fcrToCcrTable.setModel(fcrToCcrModel);
  ccrToFcrTable.setModel(ccrToFcrModel);
}

public Container getContentPane() {
  return contentPane;
```

```
    }

    /*
    void uiInit(Container container) {

//setIconImage(Toolkit.getDefaultToolkit().createImage(DemoGeneratorAppMainFrame.class.getReso
urce("[Your Icon]")));
      contentPane = (JPanel) container;
      contentPane.setLayout(null);
      //this.setSize(new Dimension(683, 554));
      //this.setTitle("Frame Title");
      fcrToCcrButton.setToolTipText("");
      fcrToCcrButton.setText("-->");
      fcrToCcrButton.setBounds(new Rectangle(332, 68, 56, 27));
      fcrToCcrButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          fcrToCcrButton_actionPerformed(e);
        }
      });
      ccrToFcrButton.setBounds(new Rectangle(331, 125, 56, 27));
      ccrToFcrButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          ccrToFcrButton_actionPerformed(e);
        }
      });
      ccrToFcrButton.setText("<--");
      ccrToFcrButton.setToolTipText("");
      //jScrollPane1.setBounds(new Rectangle(8, 18, 307, 15));
      //jScrollPane2.setBounds(new Rectangle(394, 18, 147, 18));
      //rightTable.setBorder(BorderFactory.createEtchedBorder());
      fcrToCcrTableScrollPane.setBounds(new Rectangle(7, 259, 621, 102));
      fcrToCcrTable.setBorder(BorderFactory.createEtchedBorder());
      ccrToFcrTableScrollPane.setBounds(new Rectangle(7, 370, 622, 126));
      ccrToFcrTable.setBorder(BorderFactory.createEtchedBorder());
      removeLeftToRightButton.setFont(new java.awt.Font("Dialog", 1, 12));
      removeLeftToRightButton.setText("X");
      removeLeftToRightButton.setBounds(new Rectangle(633, 297, 43, 27));
      removeLeftToRightButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          removeLeftToRightButton_actionPerformed(e);
        }
      });
      removeRightToLeftButton.setFont(new java.awt.Font("Dialog", 1, 12));
      removeRightToLeftButton.setText("X");
      removeRightToLeftButton.setBounds(new Rectangle(635, 420, 43, 27));
      removeRightToLeftButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          removeRightToLeftButton_actionPerformed(e);
        }
      });
      generateButton.setText("Generate Skeleton");
      generateButton.setBounds(new Rectangle(167, 505, 208, 27));
      generateButton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          generateButton_actionPerformed(e);
        }
      });
      fcrTreeScrollPane.setBounds(new Rectangle(9, 57, 318, 187));
      ccrTreeScrollPane.setBounds(new Rectangle(392, 56, 282, 188));
      contentPane.add(fcrToCcrTableScrollPane, null);
      contentPane.add(ccrToFcrTableScrollPane, null);
      contentPane.add(removeLeftToRightButton, null);
      contentPane.add(removeRightToLeftButton, null);
      //contentPane.add(jScrollPane1, null);
      contentPane.add(ccrToFcrButton, null);
      contentPane.add(fcrToCcrButton, null);
      //contentPane.add(jScrollPane2, null);
      contentPane.add(generateButton, null);
      contentPane.add(fcrTreeScrollPane, null);
      contentPane.add(ccrTreeScrollPane, null);
      ccrTreeScrollPane.getViewport().add(ccrTree, null);
      fcrTreeScrollPane.getViewport().add(fcrTree, null);
      //jScrollPane2.getViewport().add(rightTable, null);
      //jScrollPane1.getViewport().add(leftTable, null);
      ccrToFcrTableScrollPane.getViewport().add(ccrToFcrTable, null);
      fcrToCcrTableScrollPane.getViewport().add(fcrToCcrTable, null);
    }
    */
    void uiInit(Container container) {

//setIconImage(Toolkit.getDefaultToolkit().createImage(DemoGeneratorAppMainFrame.class.getReso
urce("[Your Icon]")));
      if (container==null)
        contentPane = new JPanel();
      else
        contentPane = (JPanel) container;


      contentPane.setLayout(new GridBagLayout());
```

```
GridBagConstraints c = new GridBagConstraints();
/*
this.setSize(new Dimension(683, 554));
this.setTitle("Frame Title");
*/
fcrToCcrButton.setToolTipText("");
fcrToCcrButton.setText("-->");
//fcrToCcrButton.setBounds(new Rectangle(332, 68, 56, 27));
fcrToCcrButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    fcrToCcrButton_actionPerformed(e);
  }
});
//ccrToFcrButton.setBounds(new Rectangle(331, 125, 56, 27));
ccrToFcrButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    ccrToFcrButton_actionPerformed(e);
  }
});
ccrToFcrButton.setText("<--");
ccrToFcrButton.setToolTipText("");
//fcrToCcrTableScrollPane.setBounds(new Rectangle(7, 259, 621, 102));
fcrToCcrTable.setBorder(BorderFactory.createEtchedBorder());
//ccrToFcrTableScrollPane.setBounds(new Rectangle(7, 370, 622, 126));
ccrToFcrTable.setBorder(BorderFactory.createEtchedBorder());
removeLeftToRightButton.setFont(new java.awt.Font("Dialog", 1, 12));
removeLeftToRightButton.setText("X");
//removeLeftToRightButton.setBounds(new Rectangle(633, 297, 43, 27));
removeLeftToRightButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    removeLeftToRightButton_actionPerformed(e);
  }
});
removeRightToLeftButton.setFont(new java.awt.Font("Dialog", 1, 12));
removeRightToLeftButton.setText("X");
//removeRightToLeftButton.setBounds(new Rectangle(635, 420, 43, 27));
removeRightToLeftButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    removeRightToLeftButton_actionPerformed(e);
  }
});
generateButton.setText("Generate Skeleton");
//generateButton.setBounds(new Rectangle(167, 505, 208, 27));
```

```
generateButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    generateButton_actionPerformed(e);
  }
});
//fcrTreeScrollPane.setBounds(new Rectangle(9, 57, 318, 187));
//ccrTreeScrollPane.setBounds(new Rectangle(392, 56, 282, 188));

c.fill = GridBagConstraints.BOTH;
c.weightx = 1.0;
c.weighty = 1.0;
c.gridwidth = 13;//4;
c.gridheight = 13;//5;
c.gridx = 1;//0;
c.gridy = 1;//0;
contentPane.add(fcrTreeScrollPane, c);

c.weightx = 0.0;
c.weighty = 0.0;
c.gridwidth = 2;//1;
c.gridheight = 2;//1;
c.gridx = 15;//5;
c.gridy = 2;//1;
contentPane.add(ccrToFcrButton, c);

c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 15;//5;
c.gridy = 6;//4;
contentPane.add(fcrToCcrButton, c);


c.fill = GridBagConstraints.BOTH;
c.weightx = 1.0;
c.weighty = 1.0;
c.gridwidth = 13;//4;
c.gridheight = 13;//5;
c.gridx = 19;//6;
c.gridy = 1;//0;
contentPane.add(ccrTreeScrollPane, c);

c.weightx = 1.0;
c.weighty = 1.0;
c.gridwidth = 22;//8;
```

```
  c.gridheight = 6;//4;
  c.gridx = 1;//0;
  c.gridy = 16;//6;
  contentPane.add(fcrToCcrTableScrollPane, c);

  c.weightx = 0.0;
  c.weighty = 0.0;
  c.gridwidth = 2;//1;
  c.gridheight = 2;//1;
  c.gridx = 24;//10;
  c.gridy = 16;//6;
  contentPane.add(removeLeftToRightButton, c);

  c.weightx = 1.0;
  c.weighty = 1.0;
  c.gridwidth = 22;//8;
  c.gridheight = 6;//4;
  c.gridx = 1;//5;
  c.gridy = 23;//12;
  contentPane.add(ccrToFcrTableScrollPane, c);

  c.weightx = 0.0;
  c.weighty = 0.0;
  c.gridwidth = 2;//1;
  c.gridheight = 2;//1;
  c.gridx = 24;//10;
  c.gridy = 23;//12;
  contentPane.add(removeRightToLeftButton, c);

  c.weightx = 1.0;
  c.gridwidth = 10;
  c.gridheight = 2;//1;
  c.gridx = 13;//5;
  c.gridy = 30;//1;
  c.insets = new Insets( 2, 10, 2, 10 );
  contentPane.add(generateButton, c);

  ccrTreeScrollPane.getViewport().add(ccrTree, null);
  fcrTreeScrollPane.getViewport().add(fcrTree, null);

  ccrToFcrTableScrollPane.getViewport().add(ccrToFcrTable, null);
  fcrToCcrTableScrollPane.getViewport().add(fcrToCcrTable, null);
}
```

```
void fcrToCcrButton_actionPerformed(ActionEvent e) {
  addMapping( fcrTree, ccrTree, fcrToCcrTable );
}

void ccrToFcrButton_actionPerformed(ActionEvent e) {
  addMapping( ccrTree, fcrTree, ccrToFcrTable );
}
void addMapping( JTree src, JTree dst, JTable mapTable ) {
  TranslationAttribute[] selectedAttrs = AttributesTreeModel.selectedAttributes(src);
  //TranslationAttribute.d(selectedAttrs);
  TranslationAttributeMapping mapping = new TranslationAttributeMapping(
                                  selectedAttrs,
                                  AttributesTreeModel.selectedAttributes(dst)[0]);
  TranslationMapTableModel mapModel = (TranslationMapTableModel) mapTable.getModel();
  mapModel.add(mapping);
}

void removeLeftToRightButton_actionPerformed(ActionEvent e) {
  removeMappings(fcrToCcrTable);
}
void removeRightToLeftButton_actionPerformed(ActionEvent e) {
  removeMappings(ccrToFcrTable);
}
void removeMappings( JTable mappingsTable ) {
  TranslationMapTableModel model=(TranslationMapTableModel)mappingsTable.getModel();
  int[] rows = mappingsTable.getSelectedRows();
  model.remove(rows);
}

void generateButton_actionPerformed(ActionEvent e) {
  StringBuffer sb = new StringBuffer(1024);
  StringBufferOutputStream out = new StringBufferOutputStream(sb);
  JavaFileDefinition fileDef = generateSkeletion(out);

  fireCodeGeneration( fileDef.getPackageName(),
                      fileDef.getPublicClass().getName(),
                      sb.toString(),
                      fileDef.targetFile() );
}

public void addCodeGenerationListener( CodeGenerationListener listener ) {
  _listeners.add( CodeGenerationListener.class, listener );
```

```
    }
    public void removeCodeGenerationListener( CodeGenerationListener listener ) {
      _listeners.remove( CodeGenerationListener.class, listener );
    }
    protected void fireCodeGeneration(  String packageName,
                                        String className,
                                        String generatedCode,
                                        File javaFile ) {
      // Guaranteed to return a non-null array
      Object[] listeners = _listeners.getListenerList();
      // Process the listeners last to first, notifying
      // those that are interested in this event
      for (int i = listeners.length-2; i>=0; i-=2) {
         EventObject event = null;
         if (listeners[i]==CodeGenerationListener.class) {
             // Lazily create the event:
             if (event == null)
                 event = new CodeGeneratedEvent(this, packageName,
                                                className, generatedCode,
                                                javaFile );
             ((CodeGenerationListener)listeners[i+1]).codeGenerated((CodeGeneratedEvent)
event);
         }
      }
    }


    /**
     * Generates the translation skeleton and writes it to the specified
     * OutputStream,
     * updates the fields _ccr, _fcrToCcrMapping and _ccrToFcrMapping.
     * Upon successful completion of this method,
     * the following tasks are completed:
     * <ol>
     * <li> Writing the translation skeleton to the specified OutputStream.
     * <li> Updating the FCR to CCR attribute mapping information contained
     *      in the _ccr field of this object.
     * <li> Updating the _fcrToCcrMapping field of this object.
     * <li> Updating the _ccrToFcrMapping field of this object.
     * </ol>
     * @param out The translation skeleton will be written to the OutputStream
     *            referenced by this parameter.
     * @return Returns the JavaFileDefinition object used to generate the
```

```
 *         skeleton.
 */
JavaFileDefinition generateSkeletion( OutputStream out ) {
  // clear the existing mappings
  _ccr.clearFCRToCCRAttributeMappings();

  JavaFileDefinition fileDef =
    TranslationFactory.newFiomTranslationFile(
                          _packageNamePrefix,
                          fcrClass.getName(),
                          ccrClass.getName()
                          );
  TranslationClassDefinition classDef
      = (TranslationClassDefinition)fileDef.getPublicClass();
  int count = fcrToCcr.getCount();
  _fcrToCcrMapping = new AttributeMapping[count];
  for (int i=0; i<count; i++) {
    classDef.addMethod(
        TranslationFactory.newFcrToCcrMethod(fcrToCcr.get(i)) );
    _fcrToCcrMapping[i] = toAttributeMapping(fcrToCcr.get(i));
    _ccr.addFCRToCCRAttributeMapping( _fcrToCcrMapping[i].fromAttributes(),
                                      _fcrToCcrMapping[i].toAttribute() );
    //p( _fcrToCcrMapping[i].toString() );
  }
  count = ccrToFcr.getCount();
  _ccrToFcrMapping = new AttributeMapping[count];
  for (int i=0; i<count; i++) {
    classDef.addMethod(
        TranslationFactory.newCcrToFcrMethod(ccrToFcr.get(i)) );
    _ccrToFcrMapping[i] = toAttributeMapping(ccrToFcr.get(i)) ;
    //p( _ccrToFcrMapping[i].toString() );
  }
  classDef.generateAbstractOverrides();

  TranslationGenerator generator = new TranslationGenerator();
  generator.generate(out, fileDef);
  return fileDef;
}
/**
 * Creates a AttributeMapping object, based on the specified
 * TranslationAttributeMapping object.
 * @param transMapping
 * @return Returns a new AttributeMapping object that reflects the
```

```
     *         information captured in the specified TranslationAttributeMapping object.
     */
    AttributeMapping toAttributeMapping ( TranslationAttributeMapping mapping ) {
      AttributeMapping result
        =                                                _fiom.factory().makeAttributeMapping(
toAttributeArray(mapping.getSourceAttributes()),
                                              toAttribute(mapping.getTargetAttribute())) ;
      return result;
    }
    /**
     * Creates an array of Attribute objects representing the
     * path information of each element of the specified array of
     * TranslationAttribute objects.
     * @param transAttr
     * @return Returns the array created.
     */
    Attribute[] toAttributeArray( TranslationAttribute[] transAttrs ) {
      Attribute[] result = new Attribute[transAttrs.length];
      for (int i=0; i<transAttrs.length; i++) {
        result[i] = toAttribute( transAttrs[i] );
      }
      return result;
    }
    /**
     * Create an Attribute object that represents the path information contained
     * in the specified TranslationAttribute object.
     * The root translation attribute is not included, since it represents the
     * actual class.
     * @param transAttr
     * @return Returns the Attribute object created.
     */
    Attribute toAttribute( TranslationAttribute transAttr ) {
      FIOMFactory f = _fiom.factory();
      TranslationAttribute[] path = transAttr.pathToArray();
      Attribute parent = null;
      Attribute curr = null;
      // start from 1, to skip the root attribute
      for (int i=1; i< path.length; i++) {
        curr = f.makeAttribute( parent, path[i].getName(),
                                f.makeTypeName(path[i].getType()) );
        parent = curr;
      }
      return curr;
    }
  } // Plugin


  public static void p(String m) {
    System.out.println(m);
  }
}


class StringBufferOutputStream extends OutputStream {
  StringBuffer _sb = null;
  /**
   *
   */
  public StringBufferOutputStream (StringBuffer sb) {
    _sb = sb;
  }
  public void write (int b) {
    StringWriter writer = new StringWriter(10);
    writer.write(b);
    _sb.append(writer.getBuffer().toString());
  }
  public void close() throws IOException {
    _sb = null;
    super.close();
  }
}
```

5. JavaFileDefinition.java

```
package mil.navy.nps.cs.oomi.translator;

import java.util.*;
import java.io.File;

/**
 * Defines a Java source file.
 * Currently, supports only one class, the only public class allowed in a
 * Java file.
 *
 * @author LSC
 * @version 1.0
 */

public class JavaFileDefinition {

  /**
   * Package name for this ClassDefinition.
   * This is optional.
   */
  protected String _packageName = null;

  /**
   * List of imports for this class.
   * The entries are strings less the keyword "import".
   */
  protected List _imports = new ArrayList(10);

  /**
   * Custom code segment for import statements managed by the user.
   */
  protected List _userManagedImports = new LinkedList();

  /**
   * The public class in this Java source file.
   */
  protected ClassDefinition _publicClass = null;

  public JavaFileDefinition( ClassDefinition publicClass ) {
    setPublicClass(publicClass);
  }

  public void setPackageName(String packageName) {
    this._packageName = packageName;
  }
  public String getPackageName() {
    return this._packageName;
  }
  /**
   * @param imports List of strings that specifies the import statements,
   *                the strings should not include the keyword "import".
   *
   */
  public void addImports( List imports ) {
    this._imports.addAll(imports);
  }

  public String[] importsToArray() {
    int size = this._imports.size();
    if (size>0)
      return (String[]) this._imports.toArray(new String[size]);
    else
      return null;
  }
  public String[] userManagedImportsToArray() {
    if (this._userManagedImports.size()>0)
      return (String[]) this._userManagedImports.toArray();
    else
      return null;
  }

  public void setPublicClass( ClassDefinition publicClass ) {
    this._publicClass = publicClass;
  }
  public ClassDefinition getPublicClass() {
    return this._publicClass;
  }
  /**
   * Returns the relative path that this object's corresponding Java source
   * file should be located.
   * For example,  for a JavaFileDefinition with main class "p1.p2.ClassA",
   * the relative path will be "p1/p2/ClassA.java".
   */
  public File targetFile() {
    String className = getPublicClass().getName();
    String path = StringHelper.replaceAll(
                              getPackageName(),
                              ".", File.separator);

    return new File(path, className + ".java");
  }

}
```

6. MethodDefinition.java

```
package mil.navy.nps.cs.oomi.translator;
```

```
import java.util.*;

/**
 * <p>
 * An instance of this class defines the source code of a method
 * of a Java class.
 *
 * <p>
 * The assumption made is that field values can only be accessed through
 * the accessor/modifier methods of the classes involved.
 *
 * @author LSC
 * @version 1.0
 */
public class MethodDefinition {

  /**
   * Name of this method.
   */
  protected String _name = null;

  /**
   * The string specifying the modifiers of this method,
   * for example, "public static".
   */
  protected String _modifiers = null;

  /**
   * List of the method's formal parameters (type and name of each parameter).
   */
  protected List _params = new ArrayList(2);
                      // assume an average of 2 parameters per method

  /**
   * Source code of the method's body.
   *
   */
  protected List _bodyLines = new LinkedList();

  /**
   * Return type of this method.
   */
```

```
protected String _returnType = null;

public MethodDefinition() {
}
public MethodDefinition(String modifiers,
                        String returnType,
                        String methodName) {
  this.setModifiers(modifiers);
  this.setReturnType(returnType);
  this.setName(methodName);
}
/**
 * Adds a parameter for this method.
 */
public void addParam(String type, String name) {
  this._params.add( new MethodParameter(type,name) );
}
/**
 * Adds parameters for this method.
 */
public void addParams(MethodParameter[] params) {
  for (int i=0; i<params.length; i++) {
    // Uses the same copy so as to maintain additional information
    // if the actual objects are descendents, like Attribute.
    this._params.add( params[i] );
  }
}

public void setName(String methodName) {
  this._name = methodName;
}
public String getName() {
  return this._name;
}
public void setModifiers(String methodModifiers) {
  this._modifiers = methodModifiers;
}
public String getModifiers() {
  return this._modifiers;
}
public void setBody( List bodyLines ) {
  this._bodyLines.clear();
  this._bodyLines.addAll(bodyLines);
  }
  public void setBody( String body ) {
    this._bodyLines.clear();
    this._bodyLines.add(body);
  }
  public String[] bodyLinesToArray() {
    if (this._bodyLines.isEmpty()) {
      return null;
    }
    else
      return (String[])
              (this._bodyLines.toArray(new String[this._bodyLines.size()]));
  }
  public MethodParameter[] paramsToArray() {
    if (this._params.isEmpty()) {
      return null;
    }
    else
      return (MethodParameter[])
            (this._params.toArray(new MethodParameter[this._params.size()]));
  }
  public void setReturnType( String returnType ) {
    this._returnType = returnType;
  }
  public String getReturnType() {
    return this._returnType;
  }
}
```

7. MethodParameter.java

```
package mil.navy.nps.cs.oomi.translator;




/**
 * @author LSC
 * @version 1.0
 */

public class MethodParameter {

  /**
 * Parameter name.
 */
protected String _name = null;

/**
 * Parameter type.
 */
protected String _type = null;

 /**
 * The class type of this attribute.
 * This is optional, used only if paramType is not
 * set in the constructor.
 */
protected Class _typeAsClass = null;

/**
 * No-argument constructor, generally not used.
 */
protected MethodParameter() {
  // do nothing
}
/**
 * Creates an instance of MethodParameter with given name and type.
 */
public MethodParameter( String paramType, String paramName ) {
  setType(paramType);
  setName(paramName);
}
/**
 * Creates an instance of MethodParameter with the given class.
 */
public MethodParameter( Class paramType, String paramName ) {
  setTypeAsClass(paramType);
  setName(paramName);
}
/**
 * Method is named setParamClass() to avoid conflict with Object.getClass().
 */
private void setTypeAsClass( Class val ) {
  _typeAsClass = val;
}
private void setName (String name) {
  _name = name;
}

/**
 * Converts the specified type into the internal format, if it is
 * not already in the format.
 *
 */
private void setType (String type) {
  _type = ReflectionHelper.toRuntimeTypeName(type);
}
/**
 * Returns the name of this parameter, with the fully qualified type name
 * converted to the source code format.
 */
public String getName() {
  return _name ;
}
public Class getTypeAsClass() {
  return _typeAsClass;
}
public String getType() {
  if (getTypeAsClass()!=null)
    return typeAsSourceCode(getTypeAsClass().getName());
  else
    return typeAsSourceCode(_type);
}
public String toString() {
  return getType() + " " + getName();
}
/**
 * A MethodParameter is decomposable only if the typeAsClass property is set.
 */
public boolean isDecomposable() {
  return getTypeAsClass()!=null;
}
/**
 * Equals if name and type matches.
 */
public boolean equals(Object obj) {
  if (obj==null)
    return false;
  else if ( ! this.getClass().isInstance(obj) )
```

```
        return false;
      else {
        MethodParameter o = (MethodParameter)obj;
        return (this.getName().equals(o.getName()) &&
                this.getType().equals(o.getType()) );
      }
    }
    /**
     * Converts the type into source code format.
     */
    public static String typeAsSourceCode(String qualifiedTypeName) {
      if (qualifiedTypeName.indexOf('[')!=-1) {
        StringBuffer result = new StringBuffer();
        String className = qualifiedTypeName;
        int len = className.length();
        int i = 0;
        while (className.charAt(i)=='[') {
          result.append("[]");
          i++;
        }
        String type;
        switch (className.charAt(i)) {
          case 'B': type = byte.class.getName(); break;
          case 'C': type = char.class.getName(); break;
          case 'D': type = double.class.getName(); break;
          case 'F': type = float.class.getName(); break;
          case 'I': type = int.class.getName(); break;
          case 'J': type = long.class.getName(); break;
          case 'S': type = short.class.getName(); break;
          case 'Z': type = boolean.class.getName(); break;
          case 'L': type = className.substring(i+1,len-1); break;
          default:
            type = "";
        }
        result.insert(0, type);
        return result.toString();
      }
      else
        return qualifiedTypeName;
    }

  }
```

8. ReflectionHelper.java

```
package mil.navy.nps.cs.oomi.translator;

/**
 * @author LSC
 * @version 1.0
 */

public class ReflectionHelper extends Exception {

  protected ReflectionHelper() {
  }

  /**
   * Converts the specified type name to the internal format that should
   * be returned from Class.getName().
   * Arrays are converted accordingly, for example,
   * "int[]" is converted to "[I",
   * "String[][]" is converted to "[[LString;".
   */
  public static String toRuntimeTypeName( String typeName ) {
    if (isArray(typeName)) {
      if (typeName.indexOf('[')==0)
        return typeName; // already in correct format
      int dimCount = countDimensions(typeName);
      StringBuffer type = new StringBuffer(typeName);
      StringHelper.replaceAll( type, "[", "" );
      StringHelper.replaceAll( type, "]", "" );
      //type = "int";
      //return String.valueOf(type.toString().equals("int"));
      //return "-" + type + "-";
      return encodeToArraySignature( type.toString(), dimCount );
    }
    else
      return typeName;
  }

  /**
   * Counts the number of dimension of the specified typeName, by counting the
   * number of '[' in the string.
   * If non-array, returns 0, that is, zero dimensions.
   * typeName can be of either internal (that is, runtime) form or
   * the form in source code, for example, "int[]" or "[I;".
   *
   * @param typeName The type name whose number of dimensions is to be
   *                 determined.
   * @return Returns the number of dimensions of the specified type.
   */
  public static int countDimensions( String typeName ) {
    return StringHelper.count( typeName, '[' );
  }

  /**
   * Determines if the given type name is an array.
   * @return True if and only if typeName contains at least one '['.
   */
  public static boolean isArray(String typeName) {
    return typeName.indexOf('[')!=-1;
  }

  /**
   * Encodes the specified type name (non-array type)
   * into its qualified type name
   * as an array.
   *
   * @param qualifiedTypeName The non-array qualified type name to be the
   *                          element type of the array.
   * @param dimCount The number of dimensions the array should have.
   *
   * @return The internal Java signature of array.
   */
  public static String encodeToArraySignature(String qualifiedTypeName,
                                              int dimCount) {
    String result;
    if (qualifiedTypeName.equals(byte.class.getName()))
      result = "B";
    else if (qualifiedTypeName.equals(char.class.getName()))
      result = "C";
    else if (qualifiedTypeName.equals(double.class.getName()))
      result = "D";
    else if (qualifiedTypeName.equals(float.class.getName()))
      result = "F";
    else if (qualifiedTypeName.equals(int.class.getName()))
      result = "I";
    else if (qualifiedTypeName.equals(long.class.getName()))
```

```
      result = "J";
    else if (qualifiedTypeName.equals(short.class.getName()))
      result = "S";
    else if (qualifiedTypeName.equals(boolean.class.getName()))
      result = "Z";
    else
      result = "L" + qualifiedTypeName + ";";
    return StringHelper.repeat("[", dimCount) + result;
  }

}
```

9. StringHelper.java

```
package mil.navy.nps.cs.oomi.translator;

/**
 * @author LSC
 * @version 1.0
 */

public class StringHelper {

  /**
   * String of whitespace characters.
   */
  public static final String WHITE_SPACE = " \t";

  /**
   * Non-instantiable class with protected constructor.
   */
  protected StringHelper() {
  }

  /**
   * Returns the index of the first occurence of <code> substr </code> within
   * <code> str </code>.
   * @returns -1 if <code> substr </code>
   *          does not exist within <code> str </code>
   */
  public static int indexOf( StringBuffer str, String substr, int beginIndex ) {
    return str.toString().indexOf(substr, beginIndex);
  }
```

```
/**
 * Replaces all occurence of oldString with newString within theString.
 */
public static void replaceAll( StringBuffer theString,
                        String oldString, String newString ) {
  if (oldString.length()!=0 ) {
    int i=0;
    int inc = newString.length();
    int start = 0;
    int offsetToEnd = oldString.length();
    while (i<theString.length()) {
      start = indexOf(theString, oldString, i );
      if (start!=-1) {
        theString.replace(start, start+offsetToEnd, newString);
        // start next scan from end of replacement
        i = start+inc;
      }
      else {
        break ;
      }
    }
  }
}
/**
 * Replaces all occurence of oldString with newString within theString.
 * A convenience method that uses String instead of the more efficient
 * StringBuffer.
 */
public static String replaceAll( String theString,
                        String oldString, String newString ) {
  StringBuffer buffer = new StringBuffer(theString);
  replaceAll(buffer,oldString,newString);
  return buffer.toString();
}
/**
 * Returns a string with <code>character</code>
 * repeated <code>count</code> times.
 */
public static String repeat( char character, int count ) {
  StringBuffer result = new StringBuffer(count);
  for (int i=0; i<count; i++) {
    result.append(character);
  }
  return result.toString();
}
/**
 * Returns a string with <code>str</code>
 * repeated <code>count</code> times.
 */
public static String repeat( String str, int count ) {
  StringBuffer result = new StringBuffer(count);
  for (int i=0; i<count; i++) {
    result.append(str);
  }
  return result.toString();
}
/**
 * Counts the number of occurence of the specified character in the
 * specified string.
 */
public static int count( String str, char c ) {
  int len = str.length();
  int result = 0;
  for (int i=0; i<len; i++) {
    if (str.charAt(i)==c)
      result ++;
  }
  return result;
}
/**
 * Concatenates the given Objects into a string
 * (by calling toString() on each object), separated by separator.
 */
public static String concat( Object[] objects, String separator ) {
  if (objects==null)
    return "";
  int len = objects.length;
  if (len == 0)
    return "";
  StringBuffer result= new StringBuffer(len*10);
  result.append(objects[0].toString());
  for (int i=1; i<len; i++) {
    result.append( separator );
    result.append( objects[i].toString() );
  }
  return result.toString();
```

```
}
/**
 * Convenience function to simplify call to
 * <a href="#stripAll(StringBuffer,String)">
 *    stripAll( StringBuffer str, String charsToStrip ) </a>
 */
public static String stripAll( String str, String charsToStrip ) {
  return  stripAll(new StringBuffer(str), charsToStrip).toString();
}

/**
 * Strip all characters found in the charsToStrip string
 * from the specified string.
 *
 * @param str The string where characters are to be stripped from.
 * @param charsToStrip The string containing the characters to be stripped
 *                     from "str".
 * @return The specified StringBuffer str.
 */
public static StringBuffer stripAll( StringBuffer str, String charsToStrip ) {
  int len = str.length();
  for (int i=len-1; i>=0; i--) {
    if ( charsToStrip.indexOf(str.charAt(i))!=-1) {
      str.deleteCharAt(i);
    }
  }
  return str;
}
/**
 * Converts the first character of the given string to upper-case.
 */
static public String capitalize( String val ) {
  if (val.length()==0)
    return "";
  return Character.toUpperCase(val.charAt(0)) + val.substring(1);
}
/**
 * Converts the first character of the given string to lower-case.
 */
static public String uncapitalize( String val ) {
  if (val.length()==0)
    return "";
  return Character.toLowerCase(val.charAt(0)) + val.substring(1);
  }


  static void p(String s) {
    System.out.println(s);
  }

}
```

10. TranslationAttribute.java

```
package mil.navy.nps.cs.oomi.translator;



import java.lang.reflect.Method;
import java.beans.*;

import java.util.List;
import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.ArrayList;

import mil.navy.nps.cs.oomi.fiom.FCRInstance;
import mil.navy.nps.cs.oomi.fiom.CCRInstance;
import mil.navy.nps.cs.oomi.exceptions.*;



/**
 * Describes an attribute of a class in the FIOM.
 * An attribute, if not a Java primitive type, can be a parent to
 * other attributes, forming a tree structure that reflects the structure of
 * this non-primitive attribute.
 * <p>
 * An Attribute can be decomposed into its child attributes,
 * but the creator of this object can prevent the expansion of this object
 * into its composing attributes by setting Expandable property to false.
 * <pre>
 * isExpandable() ==> at least one child attribute,
 * </pre>
 * but having child attributes does not imply that the attribute is expandable.
 * <p>
 * An Attribute is basically immutable, except for the child attributes that
 * it is composed of.
 */
public class TranslationAttribute extends MethodParameter {

  /**
   * The parent of this TranslationAttribute.
   */
  TranslationAttribute _parent = null; // defaults to null, indicating no parent

  /**
   * All the children that composes this TranslationAttribute.
   *
   */
  protected List _children = null;  // allocated only when used

  /**
   * Determines if this TranslationAttribute should be expanded into its composite
   * TranslationAttributes.
   *  It is up to the creator of this TranslationAttribute to decide if this
TranslationAttribute
   * can be further expanded into its composite TranslationAttributes.
   */
  boolean _expandable = false; // defaults to false

  /**
   * Determines if this TranslationAttribute is an indexed attribute, corresponding
   * to a JavaBean Index Property.
   */
  boolean _indexed = false;


  private void setExpandable(boolean val) {
    _expandable = val;
  }
  protected void setIndexed(boolean v) {
    _indexed = v;
  }

  /**
   * Returns if this TranslationAttribute is decomposable into its composite attributes.
   */
  public boolean isExpandable() {
    return _expandable;
```

```
}

/**
 * No-argument constructor, generally not used.
 */
protected TranslationAttribute() {
  // do nothing.
}
/**
 * Creates a new TranslationAttribute, with a given parent.
 */
public TranslationAttribute( TranslationAttribute parent,
                  String attributeType, String attributeName
                  ) {
  super(attributeType, attributeName);
  setParent(parent);
}
public TranslationAttribute(String attributeType, String attributeName) {
  super(attributeType, attributeName);
}
/**
 *  An attribute can only have the option of Expandable only if it has the
 *  atributeType as a Class.
 */
public TranslationAttribute( TranslationAttribute parent,
                  Class attributeType, String attributeName,
                  boolean expandable) {
  super(attributeType, attributeName);
  setParent(parent);
  setExpandable(expandable);
}
/**
 *  An attribute can only have the option of Expandable only if it has the
 *  atributeType as a Class.
 */
public TranslationAttribute( TranslationAttribute parent,
                  Class attributeType, String attributeName,
                  boolean expandable,
                  boolean indexed ) {
  super(attributeType, attributeName);
  setParent(parent);
  setExpandable(expandable);
  setIndexed(indexed);
```

```
}
/**
 * Creates a copy of the given TranslationAttribute.
 */
public TranslationAttribute(TranslationAttribute attr) {
  super(attr.getType(), attr.getName());
}
/**
 * Sets the parent.
 */
protected void setParent(TranslationAttribute val) {
  this._parent = val;
  /*
  // remove from current parent, if applicable
  if (this._parent!=null) {
    this._parent.removeChild(this);
  }
  // change to the new parent
  this._parent = val;
  if (_parent!=null)
    // notify new parent of this child attribute
    this._parent.addChild(val);
  */
}
public TranslationAttribute getParent() {
  return this._parent;
}
protected List getChildren() {
  if (isExpandable()) {
    if (_children == null ) {
      TranslationAttribute[] childs = extractChildren();
      _children = Arrays.asList(childs);
    }
  }
  return _children;
}
/**
 * Adds a child to this attribute, calls the setParent() method of the child.
 */
protected void addChild(TranslationAttribute child) {
  // should not call getChildren(), will go into infinite loop
  if (_children==null) {
    _children = new ArrayList();
  }
  _children.add(child);
  child.setParent(this);
}
/**
 * Removes the given child, calls the setParent() method of the child.
 */
protected void removeChild(TranslationAttribute child) {
  if (isExpandable())
    this.getChildren().remove(child);
    child.setParent(null);
}
/**
 * Returns the child attribute at i.
 * @return Returns the child attribute at i,
 *         null if no child at i or if this Attribute is
 *         not expandable.
 */
public TranslationAttribute childAt(int i) {
  if (isExpandable())
    return (TranslationAttribute)getChildren().get(i);
  else
    return null;
}
/**
 * Returns an Iterator for the children.
 * @return Returns an Iterator for the children,
 *         null if this TranslationAttribute is not expandable.
 */
public Iterator childrenIterator() {
  if (isExpandable())
    return this.getChildren().iterator();
  else
    return null;
}

/**
 * Inspects the given class and extracts all the attributes.
 * Attributes are defined as having public
 * accessors with names like get{AttributeName} and
 * public mutators with names like set{AttributeName}.
 * The fourth character of the method must be an uppercase letter.
 * <p>For example, the following class :
```

```
 * <pre>
 * class ClassA {
 *  public type getAtt1() {...}
 *  public type setAtt1(p0) {...}
 *  public type getAtt2() {...}
 *  public type setAtt2(p0) {...}
 * }
 * </pre>
 * will have attributes: Att1, Att2.
 * <p>
 * <b> NOTE: Assumption made is that any get/set will be matched by a corresponding
 * get/set, this method currently does not check if there is an actual match
 * and will only process all the "get" methods without parameters.
 * </b>
 * <p>
 * <b> NOTE: The getClass() method is excluded.</b>
 * <p>
 * Descendants may override this method or extractAttributeName()
 * to change the extraction logic.
 * <p>
 * A class may have attributes whose type is that of the containing
 * class, that is, cyclic references may occur.  For example,
 * <pre>
 *   class AClass {
 *    String s1;
 *    int int1;
 *    Class aClass;
 *   }
 * </pre>
 * @return Returns an array of the TranslationAttribute instances if attributes are
 *         found, else returns null.
 * @see #extractAttributeName(String)
 */
protected TranslationAttribute[] extractChildren(  )  {
  try {
    TranslationAttribute[] children = extractAttributes(this, this.getTypeAsClass());
    return children;
  }
  catch (Exception ex) {
    // this Attribute is not expandable
    return null;
  }
}
```

```
protected TranslationAttribute[] extractChildren2(  ) {
  // list methods
  Method[] methods=getTypeAsClass().getMethods();
  int len=methods.length;
  String attributeName;
  TranslationAttribute attribute;
  List result = new ArrayList(10);
  for (int i=0; i<len; i++) {
    Method m = methods[i];
    attributeName = extractAttributeName(m.getName());
    if (attributeName!=null && m.getParameterTypes().length==0) {
      attribute = new TranslationAttribute(  this,
                                  m.getReturnType(),
                                  attributeName,
                                  isExpandableType( m.getReturnType() ) ) ;
      result.add(attribute);

      //System.out.println("Attribute: " + attribute);
    }
  }
  if (len>0)
    return (TranslationAttribute[])result.toArray(new TranslationAttribute[result.size()]);
  else
    return null;
}
/**
 * Given a method's name, determines if it is an accessor that
 * matches the get* pattern and extracts the attribute name.
 * <b>The getClass() method is excluded.</b>
 * Looks for "get" methods only, since "get" methods will provide information
 * on the type of the attribute.
 * @return Returns the attribute name if method is an accessor ("get" method),
 *          else returns null.
 */
protected String extractAttributeName( String methodName ) {
  if (methodName.length()>3) {
    if (methodName.equals("getClass"))
      return null;
    char charAt3 = methodName.charAt(3);
    if (Character.isUpperCase(charAt3)) {
      if (methodName.indexOf("get")==0 ) {
        // capitalize the fourth character
        charAt3 = Character.toUpperCase(charAt3);
        return charAt3 + methodName.substring(4);
      }
    }
  }
  return null;
}
/**
 * Determines if the given type should be expandable.
 */
protected static boolean isExpandableType( Class type ) {
  return ( FCRInstance.class.isAssignableFrom(type) ||
       CCRInstance.class.isAssignableFrom(type) );
}
/**
 * Determines if this TranslationAttribute is an indexed attribute,
 * that is, a JavaBean indexed property.
 */
public boolean isIndexed() {
  return _indexed;
}
/**
 * Inspects the given class and extracts all the attributes.
 * Attributes are defined as synonymous with JavaBeans properties,
 * the specified class should therefore conform
 * to the JavaBeans specification.
 * <p>
 * The following attributes are ignored:
 * <ol>
 * <li> has<Name>
 * <li> isValid
 * </ol>
 * <p>
 * A class may have attributes whose type is that of the containing
 * class, that is, cyclic references may occur.  For example,
 * <pre>
 *   class AClass {
 *    String s1;
 *    int int1;
 *    Class aClass;
 *   }
 * </pre>
 *
 * @return Returns an array of the TranslationAttribute instances if attributes are
```

```
   *         found, else returns an empty array.
   */
  public static TranslationAttribute[] extractAttributes( TranslationAttribute parent, Class
theClass )
                                throws mil.navy.nps.cs.oomi.exceptions.
                                            IntrospectionException {
    try {
      boolean indexed = false;
      Class attrType;
      BeanInfo beanInfo = Introspector.getBeanInfo(theClass);
      PropertyDescriptor[] propDescs = beanInfo.getPropertyDescriptors();
      List result = new ArrayList(propDescs.length-1);
      String attrName;
      for (int i=0; i<propDescs.length; i++) {
        attrName = propDescs[i].getName();
        if (attrName.equals("class")
            || attrName.equals("valid")
            )
          continue;

        indexed = propDescs[i] instanceof IndexedPropertyDescriptor;
        //p( attrName  +  "  " + propDescs[i].getPropertyType().getName());
        attrType = propDescs[i].getPropertyType();

        if (indexed) {
          if (!attrType.isArray()) {
            throw new Exception( attrName + " must be an array." );
          }
        }
        result.add(new TranslationAttribute( parent,
                           attrType, attrName,
                           isExpandableType(attrType),
                           indexed ));
      }
      return (TranslationAttribute[]) result.toArray(new TranslationAttribute[result.size()]);
    }
    catch (Exception ex) {
      throw new mil.navy.nps.cs.oomi.exceptions.IntrospectionException(ex);
    }
  }
  /**
   *  Returns the full access path from this TranslationAttribute's
   *  highest ancestor.
```

```
 */
public String path( boolean includeRoot, String separator ) {
  List path = new ArrayList(5);
  TranslationAttribute curr = this;
  while (curr!=null) {
    path.add(curr);
    curr = curr.getParent();
  }
  StringBuffer result = new StringBuffer();
  Object[] objs = path.toArray();
  TranslationAttribute attr;
  int start = (includeRoot)?objs.length-1:objs.length-2;
  attr = (TranslationAttribute) objs[start];
  result.append(attr.getName());
  for (int i=start-1; i>=0; i--) {
    attr = (TranslationAttribute) objs[i];
    result.append(separator + attr.getName());
  }
  return result.toString() ;
}
/**
 *  Returns the full access path from this TranslationAttribute's
 *  highest ancestor.
 */
public String readPath( boolean includeRoot ) {
  List path = new ArrayList(5);
  TranslationAttribute curr = this;
  while (curr!=null) {
    path.add(curr);
    curr = curr.getParent();
  }
  StringBuffer result = new StringBuffer();
  Object[] objs = path.toArray();
  TranslationAttribute attr;
  int start = (includeRoot)?objs.length-1:objs.length-2;
  for (int i=start; i>=0; i--) {
    attr = (TranslationAttribute) objs[i];
    result.append(".get" + StringHelper.capitalize(attr.getName()) + "()");
  }
  return result.toString() ;
}

/**
```

```
 * Lists this TranslationAttribute object and all its ancestors in an array.
 * @return An array that holds this object and all its ancestors ordered
 *         from greatest ancestor (0th element) down to this object (last element).
 */
public TranslationAttribute[] pathToArray() {
  List path = new ArrayList(5);
  TranslationAttribute curr = this;
  while (curr!=null) {
    path.add(curr);
    curr = curr.getParent();
  }
  TranslationAttribute[] result = new TranslationAttribute[path.size()];
  // reverse the order
  if (result.length>0) {
    int i = result.length-1;
    Iterator iter = path.iterator();
    TranslationAttribute a;
    while (iter.hasNext()) {
      a = (TranslationAttribute)iter.next();
      result[i] = a;
      i--;
    }
  }
  return result;
}
/**
 * Returns the full access path source code of the "hasser"
 * of this TranslationAttribute.
 * For example, for an attribute B in object obj, accessible by the call
 *  'obj.getA().getB()',
 *  this method returns: 'obj.getA().hasB()' if includeRoot is true,
 *  and returns '.getA().hasB()' if includeRoot is false.
public String hasserText( boolean includeRoot ) {

}
 */
/**
 * Returns the source code of the has<AttributeName> method
 * for this TranslationAttribute.
 */
public String hasMethodText() {
  return "has" + StringHelper.capitalize( this.getName() ) + "()";
}
```

```
  /**
   *  Returns the full access path to write value from this TranslationAttribute's
   *  root down to this TranslationAttribute.
   *  @param includeRoot Whether to include the root attribute.
   *  @param value The string that represents the value for the write operation.
   *  @return Returns the full access path to write value from this TranslationAttribute's
   *          root down to this TranslationAttribute.
   */
  public String writePath( boolean includeRoot, String value ) {
    List path = new ArrayList(5);
    TranslationAttribute curr = this;
    while (curr!=null) {
      path.add(curr);
      curr = curr.getParent();
    }
    StringBuffer result = new StringBuffer();
    Object[] objs = path.toArray();
    TranslationAttribute attr;
    int start = (includeRoot)?objs.length-1:objs.length-2;
    for (int i=start; i>=1; i--) {
      attr = (TranslationAttribute) objs[i];
      result.append(".get" + StringHelper.capitalize(attr.getName()) + "()");
    }
    attr = (TranslationAttribute) objs[0];
    result.append(".set" +
                  StringHelper.capitalize(attr.getName()) +
                  "(" + value + ")");
    return result.toString() ;
  }


  public static void d( TranslationAttribute[] array ) {
    for (int i=0; i<array.length; i++ ) {
      System.out.println(array[i]);
    }
  }
}
```

11. TranslationAttributeMapping.java

```
package mil.navy.nps.cs.oomi.translator;
/**
```

```
 * Holds the many-to-one mapping from attributes of
 * one class to the attribute of another class.
 */
public class TranslationAttributeMapping {


  /**
   * Source attributes.
   */
  protected TranslationAttribute _sourceAttr[];

  /**
   * Target attribute.
   */
  protected TranslationAttribute _targetAttr;

  /**
   *
   */
  public  TranslationAttributeMapping(  TranslationAttribute  source[],  TranslationAttribute
target) {
    _sourceAttr = new TranslationAttribute[source.length];
    System.arraycopy( source, 0, _sourceAttr, 0, source.length );
    _targetAttr = target;
  }


  /**
   * Returns a copy of the array of from attributes.
   */
  public TranslationAttribute[] getSourceAttributes() {
    TranslationAttribute[] result = new TranslationAttribute[_sourceAttr.length];
    System.arraycopy(_sourceAttr, 0, result, 0, _sourceAttr.length);
    return result;
  }

  /**
   * Returns the target attribute.
   */
  public TranslationAttribute getTargetAttribute() {
    return _targetAttr;
  }
  /**
   * Returns a string of the source attributes separated by the
   * given separator.
   */
  public String sourceAttributesAsString( String nestedAttrSeparator,
                                          String separator,
                                          boolean includeRoot,
                                          boolean includeType) {
    if (_sourceAttr==null)
      return "";
    int len = _sourceAttr.length;
    if (len == 0)
      return "";
    StringBuffer result= new StringBuffer(len*10);
    if (includeType)
      result.append(_sourceAttr[0].getType() + " ");
    result.append(_sourceAttr[0].path(includeRoot, nestedAttrSeparator));
    for (int i=1; i<len; i++) {
      result.append( separator );
      if (includeType)
        result.append(_sourceAttr[i].getType() + " ");
      result.append( _sourceAttr[i].path(includeRoot, nestedAttrSeparator) );
    }
    return result.toString();
  }

  /**
   * Converts to string of the form:
   * "source1_type source1_name, source.
   */


}
```

12. TranslationClassDefinition.java

```
package mil.navy.nps.cs.oomi.translator;

import java.util.List;
import java.util.Iterator;
import java.util.ArrayList;
import java.util.Set;
import java.util.HashSet;
import mil.navy.nps.cs.oomi.fiom.FCRInstance;
import mil.navy.nps.cs.oomi.fiom.CCRInstance;

/**
 * @author LSC
 * @version 1.0
 */

public class TranslationClassDefinition extends ClassDefinition {

  /**
   * The FCR class name.
   */
  protected String _fcr;

  /**
   * The CCR class name.
   */
  protected String _ccr;

  public TranslationClassDefinition(String className, String parentName,
                                   String fcrName, String ccrName) {
    super(className, parentName);
    this._fcr = fcrName;
    this._ccr = ccrName;
  }
  /**
   * Generates MethodDefinition that will override the abstract methods
   * of the base class.
   * Only call this method after all "ccrToFcr_" and "fcrToCcr_" methods
   * of this object had been defined.
   */
  public void generateAbstractOverrides() {
    String fcrName = this.getFcr();
    String ccrName = this.getCcr();
    // the toFcr() method
    MethodDefinition toFcrMethod
        = new MethodDefinition( "public", FCRInstance.class.getName(), "toFCR");
    toFcrMethod.addParam( CCRInstance.class.getName(), "ccrInstance" );
    generateBody( fcrName, toFcrMethod, "ccrToFcr_",
                  ccrName);
    // the toCcr() method
    MethodDefinition toCcrMethod
        = new MethodDefinition( "public", CCRInstance.class.getName(), "toCCR");
    toCcrMethod.addParam( FCRInstance.class.getName(), "fcrInstance" );
    generateBody( ccrName, toCcrMethod, "fcrToCcr_",
                  fcrName );

    _methods.add(toCcrMethod);
    _methods.add(toFcrMethod);
  }
  /**
   * Generates the body of either "toFCR()" or "toCCR()" methods.
   * @param target Either value of _fcr or _ccr.
   * @param method The toFCR() or toCCR() method whose body is to be generated.
   * @param prefix The "ccrToFcr_" or "fcrToCcr_" attribute translation
   *               methods' prefix to be called by the method parameter.
   */
  protected void generateBody( String target, MethodDefinition method,
                               String prefix,
                               String actualParamType ) {
    // the method contains only one parameter
    MethodParameter param = method.paramsToArray()[0];
    // The parameter will be typecast to the actual type in a local variable "obj".
    String objNamePrefix = "obj"; //param.getName() ;
    List body = new ArrayList();
    Iterator iter = _methods.iterator();
    MethodDefinition callee;
    StringBuffer calleeParamStr = new StringBuffer(100);
    MethodParameter[] calleeParams;

    // generate code to type-cast method's actual parameter to appropriate type
    body.add( actualParamType + " obj = " +
                  "(" + actualParamType + ") " + param.getName() + ";" );

    // generate code to create instance of result
    body.add( target + " result = new " + target + "();" );
    body.add("");

    List codeBody = new ArrayList(20);
    AttributeParentAllocationCodeGenerator gen = new AttributeParentAllocationCodeGenerator();
    while (iter.hasNext()) {
      calleeParamStr.setLength(0);
      callee = (MethodDefinition) iter.next() ;
      if (callee.getName().indexOf(prefix)==0) {
        calleeParams = callee.paramsToArray();

        // there should be at least one parameter
        calleeParamStr.append( objNamePrefix
            + ((TranslationAttribute)calleeParams[0]).readPath(false) );
        //calleeParamStr.append( ", " + hasParam((TranslationAttribute)calleeParams[0]) );
        for (int i=1; i<calleeParams.length; i++) {
          calleeParamStr.append( ", " + objNamePrefix
            + ((TranslationAttribute)calleeParams[i]).readPath(false) );

            /*
            + TranslationFactory.firstCharToUpper(calleeParams[i].getName())
            + "()");
            */
        }

        // sample: calleeStr = fcrToCcr_...(ccrInstance.getA.get...);
        String calleeStr = callee.getName() + "(" + calleeParamStr + ")";
        TranslationAttribute attr =
                  ((AttributeTranslationMethod)callee).getTargetAttribute();
        codeBody.add("result" + attr.writePath(false, calleeStr) + ";");
        gen.generate(attr);
      }
    }
    body.addAll(gen.code());
    body.add("");
    body.addAll(codeBody);
    // generate code to return the result
    body.add("");
    body.add("return result;");
    method.setBody(body);
  }

  /**
   * Generates the code that will allocate attributes' parents.
   */
  static class AttributeParentAllocationCodeGenerator {
    /**
     * List of code generated by this class since its creation.
     */
    List _generatedCode = new ArrayList(20);
    /**
     * Generates the code that will allocate memory for the parent of the
     * specified Attribute,
     * if it has a parent attribute, that is, attr.getParent()!=null.
     * Also generates the code for the creation of the specified Attribute's
     * higher ancestors, if such code had not already been generated.
     */
    public void generate( TranslationAttribute attr ) {
      if (attr==null)
        return;
      TranslationAttribute attrParent = attr.getParent();
      // only attributes with parent should be created, that is, attributes
      // other than the root attribute
      if (attrParent!=null) {
        // if attrParent is not the root attribute, generate the allocation
        // code for its parent
        if (attrParent.getParent()!=null) {
          String result="";
          Class attrParentClass = attrParent.getTypeAsClass();
          if ( FCRInstance.class.isAssignableFrom(attrParentClass) ||
               CCRInstance.class.isAssignableFrom(attrParentClass) )
            result = ("result" + attrParent.writePath(false,
                                "new " + attrParent.getType() + "()") + ";" );
          generate(attrParent);
          if (!lineExists(result)) {
            _generatedCode.add(result);
          }
        }
      }
    }
    public List code() {
      return _generatedCode;
    }
    public boolean lineExists(String line) {
      Iterator i = _generatedCode.iterator();
      String currLine;
      while (i.hasNext()) {
        currLine = (String)(i.next());
        if (currLine.equals(line))
          return true;
      }
      return false;
    } // class AttributeParentAllocationCodeGenerator
  } // class AttributeCreationCodeGenerator
```

```
  /**
   * Generates the source code for the has<AttributeName> parameter.
  public String hasParam(TranslationAttribute attr) {

  }
   */



/*
  * Generates code that allocates all the attributes
  * in the path of the specified attribute
  public
*/
  public String getFcr() {
    return _fcr;
  }
  public String getCcr() {
    return _ccr;
  }
}
```

13. TranslationFactory.java

```
package mil.navy.nps.cs.oomi.translator;

import java.lang.reflect.Method;
import java.util.List;
import java.util.ArrayList;
import java.util.Set;
import java.util.HashSet;
import java.beans.*;
import java.util.*;

import mil.navy.nps.cs.oomi.exceptions.*;
import mil.navy.nps.cs.oomi.fiom.AbstractTranslation;

/**
 * A factory to create various FIOM translation specific class instances.
 *
 * @author LSC
 * @version 1.0
 *
 */
```

```
public class TranslationFactory {

  /**
   * The root class for all translations.
   */
  static final Class PARENT_CLASS = AbstractTranslation.class;

  /**
   * This class should never be instantiated.
   */
  protected TranslationFactory() {
  }

  public final static String METHOD_MODIFIERS = "public";

  /**
   * Creates a new MethodDefinition instance for transforming
   * CCR attributes to a FCR attribute.
   * A copy of the ccrAttrs is made.
   * First letter of each parameter is forced to lowercase, in conformance
   * with conventional Java source code standard.
   */
  public static AttributeTranslationMethod newCcrToFcrMethod(
                                  TranslationAttribute fcrAttr,
                                  TranslationAttribute[] ccrAttrs ) {
    MethodParameter[] params = new MethodParameter[ccrAttrs.length];
    System.arraycopy( ccrAttrs, 0, params, 0, ccrAttrs.length );
    AttributeTranslationMethod result
        = new AttributeTranslationMethod( METHOD_MODIFIERS,
                              fcrAttr.getType(),
                              "ccrToFcr_" + fcrAttr.path(false,"_"),
                              fcrAttr );
    result.addParams(params);
    return result;
  }
  /**
   * Generates a MethodDefinition with signature to facilitate transforming
   * the source attributes into the target attribute.
   * A copy of the fcrAttrs is made.
   * First letter of each parameter is forced to lowercase, in conformance
   * with conventional Java source code standard.
   */
```

```
public static AttributeTranslationMethod newCcrToFcrMethod(
                            TranslationAttributeMapping attrMapping) {
  AttributeTranslationMethod result
    = newCcrToFcrMethod(attrMapping.getTargetAttribute(),
                        attrMapping.getSourceAttributes());
  return result;
}

/**
 * Creates a new MethodDefinition instance for transforming
 * FCR attributes to a CCR attribute.
 * A copy of the fcrAttrs is made.
 */
public static AttributeTranslationMethod newFcrToCcrMethod(
                                  TranslationAttribute ccrAttr,
                                  TranslationAttribute[] fcrAttrs ) {
  MethodParameter[] params = new MethodParameter[fcrAttrs.length];
  System.arraycopy( fcrAttrs, 0, params, 0, fcrAttrs.length );
  AttributeTranslationMethod result
      = new AttributeTranslationMethod( METHOD_MODIFIERS,
                              ccrAttr.getType(),
                              "fcrToCcr_" + ccrAttr.path(false,"_"),
                              ccrAttr );
  result.addParams(params);
  return result;
}
/**
 * Generates a MethodDefinition with signature to facilitate transforming
 * the source attributes into the target attribute.
 */
public static AttributeTranslationMethod newFcrToCcrMethod(
                            TranslationAttributeMapping attrMapping) {
  AttributeTranslationMethod result
    = newFcrToCcrMethod(attrMapping.getTargetAttribute(),
                        attrMapping.getSourceAttributes());
  return result;
}
/**
 * Generates attribute translation methods from FCR to CCR based
 * on the AttributeMapping array.
 */
public static AttributeTranslationMethod[] newFcrToCcrMethods (
                          TranslationAttributeMapping[] mappings ) {
```

```
  AttributeTranslationMethod[] result
        = new AttributeTranslationMethod[mappings.length];
  for (int i=0; i<mappings.length;i++) {
    result[i] = newFcrToCcrMethod(mappings[i]);
  }
  return result;
}
/**
 * Generates attribute translation methods from CCR to FCR based
 * on the AttributeMapping array.
 */
public static AttributeTranslationMethod[] newCcrToFcrMethods (
                          TranslationAttributeMapping[] mappings ) {
  AttributeTranslationMethod[] result
        = new AttributeTranslationMethod[mappings.length];
  for (int i=0; i<mappings.length; i++) {
    result[i] = newCcrToFcrMethod(mappings[i]);
  }
  return result;
}



/**
 *
 * Returns a new JavaFileDefintion instance for the given fcr and ccr classes.
 *
 */
public static JavaFileDefinition
                    newFiomTranslationFile ( String packageNamePrefix,
                                             String fcr, String ccr) {
  /*
  String className = StringHelper.replaceAll(fcr,".","_")
                    + "__" // double underscores
                    + StringHelper.replaceAll(ccr,".","_");
  */

  ParsedClassName fcrClassName = ParsedClassName.parseClassName(fcr);
  ParsedClassName ccrClassName = ParsedClassName.parseClassName(ccr);
  String className = fcrClassName.className() + "__" + ccrClassName.className();

  ClassDefinition classDef
    = new TranslationClassDefinition( className,
                           PARENT_CLASS.getName(),
                           fcr, ccr );
    JavaFileDefinition fileDef = new JavaFileDefinition(classDef);
    fileDef.setPackageName(packageNamePrefix       + "." + fcrClassName.packageName()
                                                  + "." + ccrClassName.packageName());
    MethodDefinition constructor
        = new MethodDefinition( "public", "", className );
    constructor.setBody("init( " + fcr + ".class, " + ccr + ".class );");
    classDef.addMethod(constructor);
    return fileDef;
  }

  /**
   * Makes a Class instance for a single dimension array with elements of the
   * specified qualifiedTypeName.
   */
  public static Class makeArrayClass(String qualifiedTypeName)
                                             throws ClassNotFoundException {
    return Class.forName(ReflectionHelper.encodeToArraySignature(qualifiedTypeName, 1));
  }

  /**
   * Converts the first character of the given string to lower-case.
   */
  static public String firstCharToLower( String val ) {
    if (val.length()==0)
      return "";
    return Character.toLowerCase(val.charAt(0)) + val.substring(1);
  }
  /**
   * Converts the first character of the given string to upper-case.
   */
  static public String firstCharToUpper( String val ) {
    if (val.length()==0)
      return "";
    return Character.toUpperCase(val.charAt(0)) + val.substring(1);
  }

  static void p(String s) {
    System.out.println(s);
  }
}
```

```
/**
 * This class holds the class name broken down into package name and class name.
 */
class ParsedClassName {
  /**
   * The class name.
   */
  String _className = "";
  /**
   * The package name.
   */
  String _packageName = "";
  public String packageName() {
    return _packageName;
  }
  public String className() {
    return _className;
  }

  /**
   *
   */
  public ParsedClassName(  ) {
  }

  /**
   * Parses the specified class name (not an inner class) into two parts:
   * the package name and the class name of the fully qualified class name.
   * @return A ParsedClassName object with the broken down class name.
   *
   *          <br>Example 1: if the input is "a.b.c", the result would be: {"a.b","c"}.
   *          <br>Example 2: if the input is "c", the result would be: {"","c"}.
   *          <br>Example 2: if the input is "", the result would be: {"",""}.
   */
  public static ParsedClassName parseClassName( String className ) {
    ParsedClassName result = new ParsedClassName();
    if (className.length()==0)
      return result;
    StringBuffer sb = new StringBuffer(className);
    sb.reverse();
    /*
    int len = packageName.length();
    int i = len-1;
    while (i>=0) {

      i--;
    }
    */
    StringTokenizer tok = new StringTokenizer( sb.toString(), "." );
    result._className = (new StringBuffer((String)tok.nextToken())).reverse().toString();
    int endIndex = className.length()-result.className().length()-1;
    if (endIndex > 0)
      result._packageName = className.substring( 0, endIndex );
    return result;
  }

}
```

14. TranslationGenerator.java

```
package mil.navy.nps.cs.oomi.translator;

import java.io.*;
import java.util.*;

/**
 * This is basically a general purpose code generator,
 * based on a JavaFileDefinition instance.
 *
 * @author LSC
 * @version 1.0
 *
 */

public class TranslationGenerator {


  public TranslationGenerator(  ) {
  }

  /**
   * Generates the Java file into outStream.
   */
  public void generate ( OutputStream outStream,
                         JavaFileDefinition javaFileDef ) {
    Indent indent = new Indent("  ");
```

```
      // indent string is double-space characters
  PrintStream writer = new PrintStream(outStream);
  GenerationInfo genInfo = new GenerationInfo( writer,
                                               javaFileDef,
                                               indent );
  writePackageName( genInfo );
    genInfo.writer.println();
  writeImports( genInfo );
    genInfo.writer.println();
  writeClass( genInfo );
}

/**
 * Inner class for holding write information.
 */
protected class GenerationInfo {
  PrintStream writer;
  JavaFileDefinition javaFileDef;
  Indent indent;
  public GenerationInfo(PrintStream writer,
                                 JavaFileDefinition javaFileDef,
                                 Indent indent) {
    this.writer = writer;
    this.javaFileDef = javaFileDef;
    this.indent = indent;
  }
}


/**
 * Writes the given statement with indent to the Writer
 * and appends a semi-colon.
 */
protected void writeStatement(GenerationInfo genInfo, String statement) {
  genInfo.writer.print(genInfo.indent + statement + ";");
}
/**
 * Writes the given statement with indentation
 * to the Writer and appends a semi-colon and
 * a newline.
 */
protected void writeStatementLine(GenerationInfo genInfo, String statement) {
  genInfo.writer.println(genInfo.indent + statement + ";");
```

```
}

final String MARKUP_BEGIN   = "//%% begin";
final String MARKUP_END     = "//%% end";

/**
 * Marks the user defined import section.
 */
final String MARKUP_USERIMPORT  = "import";
/**
 * Marks the user defined fields section.
 */
final String MARKUP_USERFIELDS = "fields";
/**
 * Marks the user defined methods section.
 */
final String MARKUP_USERMETHODS = "methods";
/**
 * Marks the method body section.
 */
final String MARKUP_METHODBODY = "method_body";

/**
 * Writes a begin markup for tool processing.
 */
protected void writeMarkupBegin( GenerationInfo genInfo, String tagName ){
  genInfo.writer.println( genInfo.indent + MARKUP_BEGIN + " " + tagName );
}

/**
 * Writes a begin markup for tool processing, with comment after the markup.
 */
protected void writeMarkupBegin( GenerationInfo genInfo,
                                 String tagName, String comment ){
  genInfo.writer.println( genInfo.indent + MARKUP_BEGIN + " " + tagName
                          + " -- " + comment);
}

/**
 * Writes a end markup for tool processing.
 */
protected void writeMarkupEnd( GenerationInfo genInfo, String tagName ){
  genInfo.writer.println( genInfo.indent + MARKUP_END + " " + tagName );
```

```
}
/**
 * If stringArray == null, nothing is written.
 */
protected void writeStringArray( PrintStream writer, Indent indent,
                                 String[] stringArray ) {
  if (stringArray!=null) {
    for (int i=0; i<stringArray.length; i++) {
      writer.println( indent + stringArray[i] );
    }
  }
}
protected void writePackageName( GenerationInfo genInfo ) {
  if ( genInfo.javaFileDef.getPackageName()!=null ) {
    writeStatementLine( genInfo,
                        "package " + genInfo.javaFileDef.getPackageName());
  }
}
protected void writeImports( GenerationInfo genInfo ) {
  String imports[] = genInfo.javaFileDef.importsToArray();
  if (imports!=null) {
    for (int i=0; i<imports.length; i++) {
      writeStatementLine(genInfo, "import " + imports[i]);
    }
  }
  genInfo.writer.println();
  // User managed imports, each string already includes the keyword "import".
  imports = genInfo.javaFileDef.userManagedImportsToArray();
  writeMarkupBegin(genInfo, MARKUP_USERIMPORT);
  genInfo.writer.println("// -- Enter your additional imports here.");
  if (imports!=null) {
    for (int i=0; i<imports.length; i++) {
      writeStatementLine(genInfo, imports[i]);
    }
  }
  writeMarkupEnd(genInfo, MARKUP_USERIMPORT);
}
protected void writeClass( GenerationInfo genInfo ) {
  ClassDefinition classDef = genInfo.javaFileDef.getPublicClass();
  // start the class declaration
  genInfo.writer.println( "public class "
                        + classDef.getName()
                        + " extends "
                        + classDef.getParentName()
                        + " {");
  genInfo.indent.increaseIndent();
  writeMarkupBegin(genInfo, MARKUP_USERFIELDS);
  genInfo.writer.println(genInfo.indent + "// -- Enter your custom fields here.");
  writeStringArray( genInfo.writer, genInfo.indent,
                    classDef.userManagedFieldsToArray() );
  writeMarkupEnd(genInfo, MARKUP_USERFIELDS);
  genInfo.writer.println();
  writeMethods( genInfo );
  // marks the section for free-text entry for user-defined methods
  writeMarkupBegin(genInfo, MARKUP_USERMETHODS);
  genInfo.writer.println(genInfo.indent + "// -- Enter your custom methods here.");
  writeMarkupEnd(genInfo, MARKUP_USERMETHODS);

  genInfo.indent.decreaseIndent();
  // closing curly brace for the class
  genInfo.writer.println( "}" );
}

protected void writeMethods( GenerationInfo genInfo ) {
  MethodDefinition methods[]
                    = genInfo.javaFileDef.getPublicClass().methodsToArray();
  if (methods!=null) {
    for (int i=0; i<methods.length; i++) {
      writeMethod(genInfo, methods[i]);
      genInfo.writer.println();
    }
  }
}
protected void writeMethod( GenerationInfo genInfo,
                            MethodDefinition method ) {
  String returnType = method.getReturnType();
  if (returnType==null)
    returnType="";
  genInfo.writer.println( genInfo.indent
                      + method.getModifiers()
                      + " "
                      + returnType
                      + " "
                      + method.getName()
                      + " ( "
                      + methodParameters(genInfo, method.paramsToArray())
```

```
                        + " ) {" );
    genInfo.indent.increaseIndent();
    writeMarkupBegin(genInfo, MARKUP_METHODBODY, "Enter method body below:");
    writeStringArray( genInfo.writer, genInfo.indent,
                      method.bodyLinesToArray());
    writeMarkupEnd(genInfo, MARKUP_METHODBODY);
    genInfo.indent.decreaseIndent();
    genInfo.writer.println( genInfo.indent + "}" );
  }
  /**
   * If params==null, empty string is returned.
   */
  protected String methodParameters( GenerationInfo genInfo,
                                      MethodParameter params[]) {
    StringBuffer result = new StringBuffer(100);
    if (params!=null) {
      for (int i=0; i<params.length; i++) {
        if (i>0 && i!=params.length) {
          // append a comma, if not the first and not the last parameter
          result.append(", ");
        }
        MethodParameter param = params[i];
        result.append(param.getType() + " " + param.getName());
      }
      return result.toString();
    }
    else
      return "";
  }

}



/**
 * Tracks the level of indentation
 * and returns the corresponding indentation string.
 */

class Indent {

  /**
   * The current indentation string.
   */
```

```
protected StringBuffer _indent = new StringBuffer(20);

/**
 * The indent token.
 */
protected String _indentToken;

/**
 * The indent token's length.
 */
protected int _indentTokenLen = 0;

/**
 * Creates an Indent instance, with each level indented by the given
 * indent token string.
 */
public Indent( String indentToken ) {
  setIndentToken(indentToken);
}

/**
 * Increases indent level.
 */
public void increaseIndent() {
  _indent.append(getIndentToken());
}
/**
 * Decreases indent level.
 */
public void decreaseIndent() {
  int len = _indent.length();
  if (len>0) {
    _indent.delete( len-_indentTokenLen, len );
  }
}
public String getIndentToken() {
  return this._indentToken;
}
protected void setIndentToken(String token) {
  this._indentToken = token;
  this._indentTokenLen = token.length();
}
public String toString() {
    return this._indent.toString();
  }

}
```

15. TranslationMap.java

```
package mil.navy.nps.cs.oomi.translator;

import java.util.*;

/**
 * Keeps a mapping of attributes between two classes.
 */
public class TranslationMap {

  /**
   * The name of the source class.
   */
  protected String _sourceClassName;

  /**
   * The name of the target class.
   */
  protected String _targetClassName;

  /**
   * The attribute mappings.
   */
  protected List _mappingList = new LinkedList();

  /**
   * Creates a new instance of mapping from attributes of sourceClass to
   * attributes of targetClass.
   */
  public TranslationMap( String sourceClassName, String targetClassName ) {
    this._sourceClassName = sourceClassName;
    this._targetClassName = targetClassName;
  }

  /**
   * Adds an attribute mapping.
   */
```

```
public void add( TranslationAttributeMapping mapping ) {
  _mappingList.add(mapping);
}

  //-----------------
 //  Accessors
//-----------------

/**
 * Returns null if no mapping defined.
 */
public TranslationAttributeMapping[] getMappings() {
  if (_mappingList.isEmpty())
    return null;
  else
    return (TranslationAttributeMapping[])_mappingList.toArray(
                              new TranslationAttributeMapping[_mappingList.size()]);
}
/**
 * Returns the number of mappings defined.
 */
public int getCount() {
  return _mappingList.size();
}
/**
 * Returns the mapping at index idx.
 */
public TranslationAttributeMapping get(int idx) {
  return (TranslationAttributeMapping) _mappingList.get(idx);
}
/**
 * Removes the given i-th mapping.
 */
public void remove(int i) {
  _mappingList.remove(i);
}
/**
 * Removes the mappings at the given indices.
 * If indices==null, nothing is done.
 */
public void remove(int[] indices) {
  if (indices==null)
    return;
    int[] sorted = new int[indices.length];
    System.arraycopy( indices, 0, sorted, 0, indices.length );
    Arrays.sort(sorted); // inascending order
    for (int i=sorted.length-1; i>=0; i--) {
      remove(indices[i]);
    }
  }
  public String getSourceClassName() {
    return _sourceClassName;
  }
  public String getTargetClassName() {
    return _targetClassName;
  }


}
```

16. TranslationMapTableModel.java

```
package mil.navy.nps.cs.oomi.translator;

import javax.swing.table.*;
import java.util.Arrays;

/**
 * @author LSC
 * @version 1.0
 */

public class TranslationMapTableModel extends AbstractTableModel {

  /**
   * Separator for attribute path when displayed in the table.
   */
  static final String PATH_SEP = ".";

  protected TranslationMap _map;

  protected final int FROM_COL = 0;
  protected final int TO_COL = 1;

  public TranslationMapTableModel(TranslationMap map) {
    super();
    this._map = map;
  }
  public int getColumnCount() {
    /**@todo: implement this javax.swing.table.AbstractTableModel abstract method*/
    return 2;
  }
  public Object getValueAt(int row, int col) {
    /**@todo: implement this javax.swing.table.AbstractTableModel abstract method*/
    TranslationAttributeMapping mapping = _map.get(row);
    switch (col) {
      case FROM_COL: return mapping.sourceAttributesAsString(
                                    PATH_SEP, ",", false, false);
      case TO_COL: return mapping.getTargetAttribute().path(false,PATH_SEP);
    }
    return null;
  }
  public int getRowCount() {
    /**@todo: implement this javax.swing.table.AbstractTableModel abstract method*/
    return _map.getCount();
  }
  public String getColumnName(int columnIndex) {
    switch (columnIndex) {
      case FROM_COL : return _map.getSourceClassName();
      case TO_COL : return _map.getTargetClassName();
    }
    return null;
  }
  public void add( TranslationAttributeMapping newMapping ) {
    _map.add(newMapping);
    fireTableRowsInserted(_map.getCount(), _map.getCount());
  }
  public void remove( int i ) {
    _map.remove(i);
    fireTableRowsDeleted(i,i);
  }
  /**
   * Removes the given rows.
   * If rows==null, nothing is done.
   */
  public void remove( int[] rows ) {
    int[] sorted = new int[rows.length];
    System.arraycopy( rows, 0, sorted, 0, rows.length );
    Arrays.sort(sorted); // in ascending order
```

```
    for (int i=sorted.length-1; i>=0; i--) {
      remove(sorted[i]);
      fireTableRowsDeleted(sorted[i], sorted[i]);
    }
  }

}
```

17. ZzzTest_TranslationAttribute.java

```
package mil.navy.nps.cs.oomi.translator;

import junit.framework.TestCase;
import java.util.*;

public class ZzzTest_TranslationAttribute extends TestCase
{

    public ZzzTest_TranslationAttribute(String Name_)
    {
        super(Name_);
    } //public ZzzTest_Attribute(String Name_)

    protected void setUp()
    {
    } //protected void setUp()

    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_TranslationAttribute.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)

  public void test_1() {
    TranslationAttribute a = new TranslationAttribute( "int", "attribute_a" );
    TranslationAttribute a_1 =  new TranslationAttribute( a, "float", "attribute_a_1" );
    assertEquals( a, a_1.getParent() );
  }
  public void test_2() {
```

```
  TranslationAttribute a = new TranslationAttribute( "int", "attribute_a" );
  assertEquals( null, a.getParent());
}
public void test_3() {
  TranslationAttribute a = new TranslationAttribute( "int", "attribute_a" );
  assertEquals( ".getAttribute_a()", a.readPath(true));
}
public void test_4() {
  TranslationAttribute a = new TranslationAttribute( "int", "attribute_a" );
  TranslationAttribute a_1 =  new TranslationAttribute( a, "float", "attribute_a_1" );
  assertEquals( ".getAttribute_a().getAttribute_a_1()" ,
              a_1.readPath(true) );
}
public void test_5() {
  TranslationAttribute a = new TranslationAttribute( "int", "attribute_a" );
  TranslationAttribute a_1 =  new TranslationAttribute( a, "float", "attribute_a_1" );
  assertEquals( ".getAttribute_a_1()" ,
              a_1.readPath(false) );
}
public void test_6() {
  TranslationAttribute a = new TranslationAttribute( "int", "attribute_a" );
  TranslationAttribute a_1 =  new TranslationAttribute( a, "float", "attribute_a_1" );
  assertEquals( ".getAttribute_a().setAttribute_a_1(1234)" ,
              a_1.writePath(true, "1234") );
}
public void test_7() {
  TranslationAttribute a = new TranslationAttribute( "int", "attribute_a" );
  TranslationAttribute a_1 =  new TranslationAttribute( a, "float", "attribute_a_1" );
  assertEquals( ".setAttribute_a_1(1234)" ,
              a_1.writePath(false, "1234") );
}
public void test_8() {
  TranslationAttribute a = new TranslationAttribute( "int", "attribute_a" );
  TranslationAttribute a_1 =  new TranslationAttribute( a, "float", "attribute_a_1" );
  assertEquals( "attribute_a_1" ,
              a_1.path(false, "/") );
}
public void test_9() {
  TranslationAttribute a = new TranslationAttribute( "int", "attribute_a" );
  TranslationAttribute a_1 =  new TranslationAttribute( a, "float", "attribute_a_1" );
  assertEquals( "attribute_a/attribute_a_1" ,
              a_1.path(true, "/") );
}
```

```
  public void test_pathToArray_1() {
    TranslationAttribute a = new TranslationAttribute( "int", "attribute_a" );
    TranslationAttribute a_1 = new TranslationAttribute( a, "float", "attribute_a_1" );
    TranslationAttribute a_1_1 = new TranslationAttribute( a_1, "float", "attribute_a_2" );
    TranslationAttribute[] arr = { a, a_1, a_1_1 };
    assertTrue( java.util.Arrays.equals( arr, a_1_1.pathToArray() ) );
  }
  public void test_pathToArray_2() {
    TranslationAttribute a = new TranslationAttribute( "int", "attribute_a" );
    TranslationAttribute a_1 = new TranslationAttribute( a, "float", "attribute_a_1" );
    TranslationAttribute a_1_1 = new TranslationAttribute( a_1, "float", "attribute_a_2" );
    TranslationAttribute[] arr1 = { a };
    TranslationAttribute[] arr2 = { a, a_1 };
    assertTrue( Arrays.equals( arr1, a.pathToArray() )
                && Arrays.equals( arr2, a_1.pathToArray()) );
  }

} //public class ZzzTest_TranslationAttribute extends TestCase
```

18. ZzzTest_TranslationClassDefinition.java

```
package mil.navy.nps.cs.oomi.translator;

import java.util.*;

import junit.framework.TestCase;
import mil.navy.nps.cs.oomi.fiom.FCRInstance;
import mil.navy.nps.cs.oomi.fiom.CCRInstance;

public class ZzzTest_TranslationClassDefinition extends TestCase
{

    public ZzzTest_TranslationClassDefinition(String Name_)
    {
        super(Name_);
    } //public ZzzTest_TranslationClassDefinition(String Name_)

    protected void setUp()
    {
    } //protected void setUp()

    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_TranslationClassDefinition.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)

  public String stripWhiteSpace(String s) {
    return StringHelper.stripAll( s, " \t\n" );
  }
  /**
   * One-level
   */
  public void test_attribute_creation_code_1A() {
    TranslationAttribute a
        = new TranslationAttribute( null, FCRInstance_1.class, "a", true );
    TranslationAttribute a_f
        =  new TranslationAttribute( a, FCRInstance_1_1.class, "f", true );
    TranslationAttribute a_f_ff
        =  new TranslationAttribute( a_f, FCRInstance_1_1_1.class, "ff", true );
    TranslationAttribute a_f_i1
        =  new TranslationAttribute( a_f, int.class, "i1", true );
    TranslationClassDefinition.AttributeParentAllocationCodeGenerator
      generator = new TranslationClassDefinition.AttributeParentAllocationCodeGenerator();
    generator.generate(a_f_i1);
    String[]              expected              =              {              "result.setF(new
mil.navy.nps.cs.oomi.translator.FCRInstance_1_1());" };
    String[] lines = (String[]) generator.code().toArray(new String[generator.code().size()]);
    assertTrue( Arrays.equals(expected,lines) );
  }
  /**
   * One-level
   */
  public void test_attribute_creation_code_1B() {
    TranslationAttribute a
        = new TranslationAttribute( null, FCRInstance_1.class, "a", true );
    TranslationAttribute a_1
        =  new TranslationAttribute( a, FCRInstance_1_1.class, "f", true );
    TranslationAttribute a_1_1
        =  new TranslationAttribute( a_1, FCRInstance_1_1_1.class, "ff", true );
    TranslationClassDefinition.AttributeParentAllocationCodeGenerator
      generator = new TranslationClassDefinition.AttributeParentAllocationCodeGenerator();
    generator.generate(a_1_1);
    String[]              expected              =              {              "result.setF(new
mil.navy.nps.cs.oomi.translator.FCRInstance_1_1());" };
    String[] lines = (String[]) generator.code().toArray(new String[generator.code().size()]);
    assertTrue( Arrays.equals(expected,lines) );
  }
  /**
   * Two-levels.
   */
  public void test_attribute_creation_code_two_levels_1() {
    TranslationAttribute a
        = new TranslationAttribute( null, FCRInstance_1.class, "a", true );
    TranslationAttribute a_f
        =  new TranslationAttribute( a, FCRInstance_1_1.class, "f", true );
    TranslationAttribute a_f_ff
        =  new TranslationAttribute( a_f, FCRInstance_1_1_1.class, "ff", true );
    TranslationAttribute a_f_ff_i1
        =  new TranslationAttribute( a_f_ff, int.class, "i1", true );
    TranslationAttribute a_f_ff_f1
        =  new TranslationAttribute( a_f_ff, float.class, "f1", true );
    TranslationClassDefinition.AttributeParentAllocationCodeGenerator
      generator = new TranslationClassDefinition.AttributeParentAllocationCodeGenerator();
    generator.generate(a_f_ff_i1);
    generator.generate(a_f_ff_f1);
    //assertEquals( stripWhiteSpace("abc d\ne f"), stripWhiteSpace("a  \tbcdef") );
    String[] lines = (String[]) generator.code().toArray(new String[generator.code().size()]);
    String[] expected = new String[] {
        "result.setF(new mil.navy.nps.cs.oomi.translator.FCRInstance_1_1());",
        "result.getF().setFf(new mil.navy.nps.cs.oomi.translator.FCRInstance_1_1_1());"
      };
    assertTrue( Arrays.equals(expected, lines) );
  }
  /**
   * Two-levels.
   */
  public void test_attribute_creation_code_two_levels_2() {
    TranslationAttribute a
        = new TranslationAttribute( null, FCRInstance_1.class, "a", true );
    TranslationAttribute a_f
        =  new TranslationAttribute( a, FCRInstance_1_1.class, "f", true );
    TranslationAttribute a_f_d1
        =  new TranslationAttribute( a_f, double.class, "d1", true );
    TranslationAttribute a_f_ff
```

```
        =  new TranslationAttribute( a_f, FCRInstance_1_1_1.class, "ff", true );
    TranslationAttribute a_f_ff_i1
        =  new TranslationAttribute( a_f_ff, int.class, "i1", true );
    TranslationAttribute a_f_ff_f1
        =  new TranslationAttribute( a_f_ff, float.class, "f1", true );
    TranslationClassDefinition.AttributeParentAllocationCodeGenerator
      generator = new TranslationClassDefinition.AttributeParentAllocationCodeGenerator();
    generator.generate(a_f_ff_i1);
    generator.generate(a_f_ff_f1);
    generator.generate(a_f_d1);

    String[] lines = (String[]) generator.code().toArray(new String[generator.code().size()]);
    String[] expected = new String[] {
        "result.setF(new mil.navy.nps.cs.oomi.translator.FCRInstance_1_1());",
        "result.getF().setFf(new mil.navy.nps.cs.oomi.translator.FCRInstance_1_1_1());"
      };
    assertTrue( Arrays.equals(expected, lines) );
  }

} //public class ZzzTest_TranslationClassDefinition extends TestCase

class FCRInstance_1 extends FCRInstance {
  FCRInstance_1_1 f;
}
class FCRInstance_1_1 extends FCRInstance {
  int i1;
  double d1;
  FCRInstance_1_1_1 ff;
}
class FCRInstance_1_1_1 extends FCRInstance {
  int i1;
  float f1;
}
```

19. ZzzTest_TranslationFactory.java

```
package mil.navy.nps.cs.oomi.translator;

import junit.framework.TestCase;
import java.util.Arrays;
import mil.navy.nps.cs.oomi.exceptions.IntrospectionException;

public class ZzzTest_TranslationFactory extends TestCase
```

```
{

    public ZzzTest_TranslationFactory(String Name_)
    {
        super(Name_);
    } //public ZzzTest_TranslatorFactory(String Name_)

    protected void setUp()
    {
    } //protected void setUp()

    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_TranslationFactory.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)

  public void testExtractAttributes() throws IntrospectionException {
    TranslationAttribute[]
      oracle = {new TranslationAttribute("java.lang.String", "string1"),
                new TranslationAttribute("java.lang.String", "string2") };
    assertTrue( Arrays.equals(
                  TranslationAttribute.extractAttributes(
                                            null,
                                            TestAttributeExtraction.class),
                  oracle)
               );
  }
  public void testParseClassName_1() {
    ParsedClassName c = ParsedClassName.parseClassName("a.b.c");
    assertEquals( "c", c.className() );
  }
  public void testParseClassName_2() {
    ParsedClassName c = ParsedClassName.parseClassName("a.b.c");
    assertEquals( "a.b", c.packageName() );
  }
  public void testParseClassName_3() {
    ParsedClassName c = ParsedClassName.parseClassName("c");
    assertEquals( "", c.packageName() );
  }
  public void testParseClassName_4() {
    ParsedClassName c = ParsedClassName.parseClassName("c");
    assertEquals( "c", c.className() );
  }
  public void testParseClassName_5() {
    ParsedClassName c = ParsedClassName.parseClassName("");
    assertTrue( c.className().equals("") && c.packageName().equals("") );
  }
} //public class ZzzTest_TranslationFactory extends TestCase

class TestAttributeExtraction {
  private String getAString() { return null; }
  public String getString1() { return null; }
  public String aString() {return null;}
  public String getString2() { return null; };

}
```

B. PACKAGE: mil.navy.nps.cs.babel.plugin

1. CodeGeneratedEvent.java

```
package mil.navy.nps.cs.babel.plugin;

import java.io.File;
import java.util.EventObject;

/**
 * This event signals that code had been generated from the
 * translation generator.
 */
public class CodeGeneratedEvent extends EventObject {

  /**
   * The source code generated.
   */
  private String _sourceCode = "";
```

```
/**
 * The package name of the generated code.
 */
private String _packageName = "";

/**
 * The relative file path for the generated code's file.
 */
private File _javaFile = null;

/**
 * The class name of the generated translation.
 */
private String _className = "";

public CodeGeneratedEvent(Object source,
                          String packageName, String className,
                          String sourceCode,
                          File javaFile) {
  super(source);
  this._className = className;
  this._packageName = packageName;
  this._sourceCode = sourceCode;
  this._javaFile = javaFile;
}

/**
 * Returns the source code generated.
 */
public String getSourceCode() {
  return _sourceCode;
}
/**
 * Returns the package name of the generated code.
 */
public String getPackageName() {
  return _packageName;
};

/**
 * Returns the class name of the generated translation.
 */
public String getClassName() {
    return _className;
  };

  /**
   * Returns the file path (relative) that the Java file to contain generated code.
   * For example,  for generated code whose main class is "p1.p2.ClassA",
   * the relative path will be "p1/p2/ClassA.java" or "p1\p2\ClassA.java",
   * depending on the OS.
   */
  public File getJavaFile() {
    return _javaFile;
  }
}
```

2. CodeGenerationListener.java

```
package mil.navy.nps.cs.babel.plugin;

import java.util.EventListener;

/**
 * A plugin for the Translation Generator to interface with the Babel IDE.
 */
public interface CodeGenerationListener
                    extends EventListener {

  /**
   * Called by the TranslationGeneratorPlugin when code is generated.
   * @param event The event generated, contains information on
   *              the code generated, its package name, class name and the
   *              relative path of the Java file for the generated code.
   * @see CodeGeneratedEvent
   */
  public void codeGenerated( CodeGeneratedEvent event );

}
```

3. TranslationGeneratorPlugin.java

```
package mil.navy.nps.cs.babel.plugin;

import java.awt.Container;

/**
```

```
 * A plugin for the Translation Generator to interface with the Babel IDE.
 */
public interface TranslationGeneratorPlugin {

  /**
   * Returns the container that this object is being displayed in.
   *
   * @return Returns the Container object that the UI components of this object
   *          is displayed in.
   */
  public Container getContentPane();

  /**
   * Adds the code generation listener.
   */
  public void addCodeGenerationListener( CodeGenerationListener listener );

  /**
   * Removes the specified CodeGenerationListener.
   */
  public void removeCodeGenerationListener( CodeGenerationListener listener );


}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. TRANSLATOR SOURCE CODE

A. PACKAGE: mil.navy.nps.cs.oomi.translator

1. Translator.java

```
package mil.navy.nps.cs.oomi.translator;

import java.net.URL;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.InputStreamReader;
import java.util.*;
import java.lang.reflect.InvocationTargetException;

import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;



/**
 * The OOMI translator.
 * The translator uses one specific instance of a FIOM.
 *
 * <p>
 * Typical translation scenario:
 * <img src="doc-files/Translator_usage_scenario_seq.gif">
 *
 * @author LSC
 * @version 1.0
 */

public class Translator {

  /**
   * The Federation that this Translator is operating on.
   */
  protected FIOM _fiom = null;
```

```
/**
 * Creates a translator that operates on the specified federation (fiom).
 */
public Translator( FIOM fiom ) {
  setFIOM(fiom);
}

public FIOM getFIOM() {
  return _fiom;
}
public void setFIOM(FIOM fiom) {
  _fiom = fiom;
}




/**
 * Translates the given CCRSchema instance into its corresponding
 * FCRSchema instance.
public FCRSchema toFCRInstance ( CCRSchema ccrInstance ) {
  return null;
}
 */

/**
 * Translates the given FCRSchema instance into its corresponding
 * CCRSchema instance.
public CCRSchema toCCRInstance ( FCRSchema fcrInstance,
                                 String destinationSystem ) {
  return null;
}
 */

/**
 * Translates the given CCRSchema instance into a FCRSchema instance
 * for the given targetSystem.
 *
 * @return A FCRSchema instance translated from the given CCRSchema instance.
public FCRSchema toDestinationFCRInstance ( FCRSchema fcrInstance,
                                    String destinationSystem ) {
  return null;
}
 */
```

```
/**
 * Creates the TranslatorSourceInfo object based on the specified information.
 * @param systemName      The source system name.
 * @param xmlNameSpaceURI The namespace URI for the source XML document.
 *                        Generally, this information should be present in
 *                        the XML document received or can be supplied by the
 *                        sender in the communication protocol.
 * @param xmlStream       The input stream supplying the source
 *                        XML document content.
 * @return An instance of TranslatorSourceInfo.
 * @throws UnknownNameSpaceURIException When the specified URI is not known
 *                                       to the current Federation (FIOM).
 */
public TranslatorSourceInfo createSourceInfo( String systemName,
                                              URL xmlNameSpaceURI,
                                              InputStream xmlStream )
                                throws UnknownNameSpaceURIException,
                                       UnmarshalException,
                                       ClassNotFoundException,
                                       InstantiationException,
                                       IllegalAccessException   {
  TranslatorSourceInfo result = new TranslatorSourceInfo(systemName);

  CCR ccr = getFIOM().findCCR( systemName, xmlNameSpaceURI );
  if (ccr == null)
    throw new UnknownNameSpaceURIException(xmlNameSpaceURI);
  FCR fcr = ccr.getFCR();

  // Load the CCR's Java class for unmarshalling.
  Class ccrClass = Class.forName( ccr.getJavaClassName() );

  // Calls the unmarshaller.
  CCRInstance ccrInstance = CCRInstance.unmarshal( ccrClass, xmlStream );

  AbstractTranslation translation
    = findTranslation( Class.forName(fcr.getJavaClassName()),
                       ccrClass );
  result.setCCR(ccr);
  result.setCCRInstance(ccrInstance);
  result.setFCRInstance( translation.toFCR(ccrInstance) );
  return result;
} // createSourceInfo(String, URL)

/**
 * Transforms the specified fcrInstance into its CCRInstance of the
 * specified CCR.
 */
public CCRInstance toCCRInstance( CCR ccr,
                                  FCRInstance fcrInstance )
                                  throws ClassNotFoundException,
                                         IllegalAccessException,
                                         InstantiationException {
  Class ccrInstanceClass = Class.forName(ccr.getJavaClassName());
  AbstractTranslation translation
    = findTranslation( fcrInstance.getClass(),
                       ccrInstanceClass );
  CCRInstance ccrInstance = translation.toCCR(fcrInstance);

  return ccrInstance;
}

/**
 * Finds the Translation for the specified FCR and CCR.
 */
public AbstractTranslation findTranslation( Class fcrInstanceClass,
                                            Class ccrInstanceClass )
                              throws ClassNotFoundException,
                                     IllegalAccessException,
                                     InstantiationException {
  String className = getFIOM().findTranslation(
                                   fcrInstanceClass.getName(),
                                   ccrInstanceClass.getName()  );
  AbstractTranslation result
    = (AbstractTranslation) ( Class.forName( className ).newInstance() );

  return result;
}

/**
 * Finds the possible destinations for the given TranslatorSourceInfo.
 * @param source
 * @param destinationSystem The destination system.
 * @return Returns an array of CCR that are suitable destinations based
 *         on the given source and destinationSystem. The array is sorted
 *         from least suitable (element 0) to most suitable (last element).
 *         Suitability is defined as the count of the number optional
 *         attributes satisfied by the source's FCR instance.
 */
public CCR[] findDestinations( TranslatorSourceInfo source,
                               String destinationSystem )
                               throws InvocationTargetException,
                                      IllegalAccessException,
                                      java.beans.IntrospectionException,
                                      NoSuchMethodException {
  FCR srcFCR = source.getFCR();
  FEV fev = srcFCR.getFEV();
  FE  fe  = fev.getFE();
  /*
  mil.navy.nps.cs.oomi.fiom.Attribute[] mandatoryAttrs
      = source.getCCR().mandatoryFCRAttributes();
  int mandatoryCount = mandatoryAttrs.length;
  */
  List candidates
    = new ArrayList( Arrays.asList(fe.findCCR( destinationSystem ) ) );
  ListIterator iter = candidates.listIterator();

  String attrNames[];
  Attribute[] srcFCRAttrs = srcFCR.getFCRSchema().enumerateAttributes();

  List matchCounts = new ArrayList(10);
  while (iter.hasNext()) {
    CCR dstCCR = (CCR)iter.next();
    MatchCount count = attributeMatchCount( source, dstCCR );
    if (count.mandatorySatisfied())
      matchCounts.add(count);
  }
  MatchCount[] sortedCounts
    = (MatchCount[]) matchCounts.toArray(new MatchCount[matchCounts.size()]);
  Arrays.sort( sortedCounts );
  CCR[] result = new CCR[sortedCounts.length];
  for (int i=0; i<sortedCounts.length; i++) {
    result[i] = sortedCounts[i].getCCR();
  }
  return result;
}
```

```
/**
 * A convenience method to perform a one-step translation from
 * the source XML document to the destination XML document.
 *
 * @return Returns true if the source XML document is successfully translated
 *         to the destination system.
 *         Returns false if no matching destination CCR can be found.
 */
public boolean translate( String sourceSystemName, URL sourceNamespaceURI,
                       InputStream sourceStream,
                       String destinationSystemName,
                       OutputStream destinationStream )
                       throws ClassNotFoundException,
                              UnmarshalException,
                              UnknownNameSpaceURIException,
                              java.beans.IntrospectionException,
                              IllegalAccessException,
                              java.lang.InstantiationException,
                              java.lang.reflect.InvocationTargetException,
                              NoSuchMethodException,
                              org.exolab.castor.xml.ValidationException,
                              org.exolab.castor.xml.MarshalException {
  //-- Translates into the source FCRInstance
  TranslatorSourceInfo srcInfo =
    createSourceInfo( sourceSystemName, sourceNamespaceURI, sourceStream );

  //-- determine the possible destination CCRs
  //-- for the specified destination system
  CCR[] destCCR = findDestinations( srcInfo, destinationSystemName );
  if (destCCR.length==0)
    return false;

  //--
  FCRInstance srcFCRInstance = srcInfo.getFCRInstance();
  //-- use the most suitable CCR
  CCR targetCCR = destCCR[destCCR.length-1];
  //--
  AbstractTranslation dstTranslation =
      (AbstractTranslation)
      Class.forName(getFIOM().findTranslation( targetCCR.getFCR().getJavaClassName(),
                                      targetCCR.getJavaClassName() )).newInstance();
  //-- load the class definition for the target FCRInstance
  Class dstFCRInstanceClass = (Class)Class.forName(targetCCR.getFCR().getJavaClassName());
  //-- perform a shallow copy of srcFCRInstance into a new object
  FCRInstance dstFCRInstance
      = FCRInstanceHelper.clone( srcInfo.getFCRInstance(), dstFCRInstanceClass );
  //-- final translation
  CCRInstance dstCCRInstance = dstTranslation.toCCR( dstFCRInstance );
  //-- marshal to XML
  dstCCRInstance.marshal( destinationStream );

  return true;
}



/**
 *  Determines the mandatory and optional attribute match count of
 *  the specified dstFCR to that of the srcFCR.
 *  @param srcFCRInstance The source FCR instance.
 *  @param dstCCR The possible destination CCR to match against.
 *  @return Returns a MatchCount that indicates the number of mandatory and
 *          optional attributes of dstFCR that matches that of dstCCR.
 */
static MatchCount attributeMatchCount( TranslatorSourceInfo src,
                                       CCR dstCCR )
                                       throws InvocationTargetException,
                                              IllegalAccessException,
                                              java.beans.IntrospectionException,
                                              NoSuchMethodException {
  MatchCount count = new MatchCount(dstCCR);
  FCR dstFCR = dstCCR.getFCR();

  Attribute[] attrArray = dstFCR.getFCRSchema().enumerateAttributes();
  Attribute attr;
  for (int i=0; i<attrArray.length; i++) {
    attr = attrArray[i];
    if (src.hasAttribute(attr)) {
      if ( src.hasValue(attr) )
        if (dstCCR.isMandatoryFCRAttribute(attr))
          count.incMandatory();
        else
          count.incOptional();
    }
  }
  count.setMandatorySatisfied(
            count.getMandatoryMatchCount() == dstCCR.mandatoryFCRAttributeCount()  );
    return count;
  }

  public static void p(String msg) {
    System.out.println(msg);
  }
}



/**
 * A class to hold information on the Source required for translation.
 */
class TranslatorSourceInfo {
  String _systemName = "";
  CCR _ccr = null;
  FCRInstance _fcrInstance = null;
  CCRInstance _ccrInstance = null;
  /**
   * A map of all attributes, including all descendent attributes,
   * with the full path of the attributes as the key.
   */
  Map _fcrAttributesMap = new HashMap();

  TranslatorSourceInfo( String systemName ) {
    setSystemName(systemName);
  }

  public String getSystemName() {
    return _systemName;
  }
  public void setSystemName(String systemName) {
    _systemName = systemName;
  }
  public FCR getFCR() {
    return _ccr.getFCR();
  }
  public CCR getCCR() {
    return _ccr;
  }
  public void setCCR(CCR ccr) {
    _ccr = ccr;
  }
  public FCRInstance getFCRInstance() {
    return _fcrInstance;
  }
  public void setFCRInstance(FCRInstance v) {
    _fcrAttributesMap.clear();
    _fcrInstance = v;
  }
  public CCRInstance getCCRInstance() {
    return _ccrInstance;
  }
  public void setCCRInstance(CCRInstance v) {
    _ccrInstance = v;
  }
  /**
   * Indicates if the attribute in getFCRInstance() has any value set.
   */
  public boolean hasValue(Attribute attr)
                          throws java.beans.IntrospectionException,
                                 IllegalAccessException,
                                 InvocationTargetException,
                                 NoSuchMethodException {
    return FCRInstanceHelper.hasValue(getFCRInstance(), attr);
  }
  /**
   * Returns the FCR attribute map.
   */
  Map fcrAttributeMap() {
    if (_fcrAttributesMap.size()==0 && getFCR()!=null) {
      Attribute[] attrArray = getFCR().getFCRSchema().enumerateAttributes();
      for (int i=0; i<attrArray.length; i++) {
        _fcrAttributesMap.put( attrArray[i].fullPath(), attrArray[i] );
      }
    }
    return _fcrAttributesMap;
  }
  /**
   * Indicates if the source FCR has the specified attribute.
   */
  public boolean hasAttribute( Attribute attr ) {
    return fcrAttributeMap().containsKey(attr.fullPath());
  }
}

/**
```

```
 * A class to hold information on mandatory match and overall match counts.
 */
class MatchCount implements Comparable {
  /**
   * Number of mandatory matches.
   */
  int _mandatoryMatchCount=0;

  /**
   * Number of optional matches.
   */
  int _optionalMatchCount=0;

  /**
   * Indicates if all mandatory attributes had been satisfied.
   */
  boolean _mandatorySatisfied = false;

  /**
   * The CCR that this MatchCount is based on.
   */
  CCR _ccr;


  public MatchCount( CCR ccr ) {
    _ccr = ccr;
  };

  MatchCount( CCR ccr, boolean mandatorySatisfied,
              int mandatoryCount, int optionalCount ) {
    _ccr = ccr;
    _mandatorySatisfied = mandatorySatisfied;
    _mandatoryMatchCount = mandatoryCount;
    _optionalMatchCount  = optionalCount;
  }

  /**
   * Increase mandatory count by one.
   */
  public void incMandatory() {
    _mandatoryMatchCount++;
  }
  /**
```

```
    * Increase optional count by one.
    */
   public void incOptional() {
     _optionalMatchCount++;
   }
   public void setMandatorySatisfied( boolean v ) {
     _mandatorySatisfied = v;
   }

   public int getMandatoryMatchCount() {
     return _mandatoryMatchCount;
   }
   public int getOptionalMatchCount() {
     return _optionalMatchCount;
   }
   public boolean mandatorySatisfied() {
     return _mandatorySatisfied;
   }
   public int totalMatchCount() {
     return getMandatoryMatchCount() + getOptionalMatchCount();
   }
   public CCR getCCR() {
     return _ccr;
   }
   public int compareTo(Object obj) {
     MatchCount c = (MatchCount)obj;
     if (getMandatoryMatchCount()==c.getMandatoryMatchCount()) {
       if (getOptionalMatchCount()>c.getOptionalMatchCount())
         return 1;
       else if (getOptionalMatchCount()<c.getOptionalMatchCount())
         return -1;
       else
         return 0;
     }
     else if (getMandatoryMatchCount()>c.getMandatoryMatchCount())
       return 1;
     else // (getMandatoryMatchCount()<c.getMandatoryMatchCount())
       return -1;
   }

 }
```

2. ZzzTest_Translator.java

```
package mil.navy.nps.cs.oomi.translator;

import java.net.URL;
import java.io.*;
import java.util.*;

import junit.framework.TestCase;
import mil.navy.nps.cs.oomi.*;
import mil.navy.nps.cs.oomi.impl.*;
import mil.navy.nps.cs.oomi.fiom.*;
import mil.navy.nps.cs.oomi.exceptions.*;
import testclasses.*;

public class ZzzTest_Translator extends TestCase
{

    public ZzzTest_Translator(String Name_)
    {
        super(Name_);
    } //public ZzzTest_Translator(String Name_)

    OOMIDatabase oomiDb = new OOMIDatabaseImpl();

    protected void setUp()
    {
    } //protected void setUp()

    protected void tearDown()
    {
    } //protected void tearDown()

    public static void main(String[] args)
    {
        String[] testCaseName = {ZzzTest_Translator.class.getName()};
        junit.swingui.TestRunner.main(testCaseName);
    } //public static void main(String[] args)


  public static void p(String msg) {
    System.out.println(msg);
  }
```

```
/**
 * Builds the FIOM for testing the Translator.
 * <pre>
 *  FE_1
 *    |_ FE_1_1
 *      |_ FE_1_1_1
 *      |_ FE_1_1_2
 *  FE_2
 *    |_ FE_2_1
 *    |_ FE_2_2
 *
 *
 *  FE_1
 *    |_ rootFEV_1_A  -- FCR_1_A
 *      |_ FEV_1_A_A    -- FCR_1_A_A
 *      |_ FEV_1_A_B    -- FCR_1_A_B
 *      |_ FEV_1_A_C    -- FCR_1_A_C
 *
 *  FCR_1_A
 *    |_ ccr_1_A_1
 *    |_ ccr_1_A_2
 *
 *  FCR_1_A_A
 *    |_ ccr_1_A_A_1
 *    |_ ccr_1_A_A_2
 *
 * </pre>
 */
public static FIOM createFIOM0(OOMIDatabase oomiDb) throws Exception {
  FIOM fiom = oomiDb.newFIOM("FIOM for Testing");
  FE FE_1 = fiom.newFE("FE_1");
    FE FE_1_1 = FE_1.newChild("FE_1_1");
      FE FE_1_1_1 = FE_1_1.newChild("FE_1_1_1");
      FE FE_1_1_2 = FE_1_1.newChild("FE_1_1_2");
  FE FE_2 = fiom.newFE("FE_2");
    FE FE_2_1 = FE_1.newChild("FE_2_1");
      FE FE_2_1_1 = FE_1_1.newChild("FE_2_1_1");
      FE FE_2_1_2 = FE_1_1.newChild("FE_2_1_2");
    FEV     rootFEV_1_A     =     FE_1.createRootFEV("rootFEV_1_A");     FCR     FCR_1_A     =
rootFEV_1_A.createFCR("FCR_1_A_A");
      FEV     FEV_1_A_A     =     rootFEV_1_A.newChild("FEV_1_A_A");     FCR     FCR_1_A_A     =
FEV_1_A_A.createFCR("FCR_1_A_A");
      FEV     FEV_1_A_B     =     rootFEV_1_A.newChild("FEV_1_A_B");     FCR     FCR_1_A_B     =
FEV_1_A_B.createFCR("FCR_1_A_A");
      FEV     FEV_1_A_C     =     rootFEV_1_A.newChild("FEV_1_A_C");     FCR     FCR_1_A_C     =
FEV_1_A_C.createFCR("FCR_1_A_A");

    CCR ccr_1_A_1 = FCR_1_A.newCCR( "System", "CCR_1_A_1",
                                new URL("http://localhost/testclasses/Address.xsd") );
    CCR ccr_1_A_2 = FCR_1_A.newCCR( "System", "CCR_1_A_2",
                                new URL("http://localhost/testclasses/Customer.xsd") );

    CCR ccr_1_A_A_1 = FCR_1_A_A.newCCR( "System", "CCR_1_A_A_1",
                                new URL("http://localhost/testclasses/Invoice.xsd") );
    CCR ccr_1_A_A_2 = FCR_1_A_A.newCCR( "System", "CCR_1_A_A_2",
                                new URL("http://localhost/testclasses/Item.xsd") );



    return fiom;
  }

  /**
   * Tests : Translator(FIOM), getFIOM(), setFIOM(FIOM).
   */
  public void test_constructor_getsetFIOM_1() throws Exception {
    FIOM fiom = oomiDb.newFIOM("New FiOm");
    Translator trans = new Translator( fiom );
    assertTrue( fiom == trans.getFIOM() );
  }

  /**
   *  Tests the conversion of a XML document into its corresponding FCRInstance.
   */
  public void test_createSourceInfo() throws Exception {
    FIOM fiom = oomiDb.newFIOM("New FiOm");
    Translator trans = new Translator( fiom );
    String sourceSystemName = "TheSource";
    //URL nsURI = new URL( "http://testing.123.com/" );
    URL nsURI = new URL( (new testclasses.InvoiceDescriptor()).getNameSpaceURI() );

    fiom.addTranslation( testclasses.Item.class.getName(),
                         testclasses.Invoice.class.getName(),
                    testclasses_Item__testclasses_Invoice.class.getName() );
    CCR ccr = fiom.newFE("FE").newChild("Child")
        .createRootFEV("root FEV").createFCR("fcr")
        .newCCR( sourceSystemName, "CCR", nsURI );
    ccr.setJavaClassName( testclasses.Invoice.class.getName() );
    ccr.getFCR().setJavaClassName( testclasses.Item.class.getName() );

    String xmlString = "<?xml version=\"1.0\"?>" +
                    "<invoice        xmlns=\"http://castor.exolab.org/Test/Invoice\"><ship-
to><name>Ryan  Madden</name><address><street1>2000  Alameda  de  las  Pulgas</street1><city>San
Mateo</city><state>CA</state><zip-code>94403</zip-code></address><phone>650-345-
2777</phone></ship-to><item     Id=\"0.9.3\"     InStock=\"true\"     Category=\"Open     Source
Project\"><Quantity>1</Quantity><Price>100.00</Price></item><shipping-
method><carrier>UPS</carrier><option>Ground    Trak</option><estimated-delivery>P1D</estimated-
delivery></shipping-method><shipping-date><date>2000-10-
24</date><time>12:30:00.0</time></shipping-date></invoice>";

    ByteArrayOutputStream out = new ByteArrayOutputStream(5000);
    Writer w = new OutputStreamWriter(out);
    w.write(xmlString);
    w.flush();

    Invoice      invoice      =      Invoice.unmarshal(new      InputStreamReader(      new
ByteArrayInputStream(out.toByteArray()) ));
    String category = invoice.getShippingMethod().getCarrier();

    InputStream in = new ByteArrayInputStream( out.toByteArray() );

    TranslatorSourceInfo srcInfo =
      trans.createSourceInfo( sourceSystemName, nsURI, in );

    testclasses.Item item = (testclasses.Item) srcInfo.getFCRInstance();

    assertEquals( "category: " + category , item.getCategory() );
  }

  public void test_MatchCount_compareTo() {
    MatchCount c1 = new MatchCount( null, true, 5, 10 );
    MatchCount c2 = new MatchCount( null, true, 5, 10 );
    MatchCount c3 = new MatchCount( null, true, 5, 5 );
    MatchCount c4 = new MatchCount( null, true, 5, 15 );
    MatchCount c5 = new MatchCount( null, true, 6, 10 );
    assertTrue( c1.compareTo(c2)==0 &&
                c1.compareTo(c3)>0 &&
                c1.compareTo(c4)<0 &&
                c1.compareTo(c5)<0 );
  }

  /**
   * Builds up the test FIOM.
   * The FIOM structure is shown in the diagram below:
   * <img src="doc-files/ZzzTest_Translator_sample_FIOM_1.gif">
   *
   */
  public static void createTestFIOM( FIOM fiom ) throws DuplicateKeyException,
                                               UniqueNameViolationException,
                                               java.net.MalformedURLException {
    FIOMFactory f = fiom.factory();
    FE entity_1 = fiom.newFE( "Entity_1" );
    FEV entity_1_View_1 = entity_1.createRootFEV( "Entity_1_View_1" );
    FEV entity_1_View_1_A = entity_1_View_1.newChild( "Entity_1_View_1_A" );
    FEV entity_1_View_1_B = entity_1_View_1.newChild( "Entity_1_View_1_B" );
    FEV entity_1_View_1_C = entity_1_View_1.newChild( "Entity_1_View_1_C" );
    FCR entity_1_View_1_fcr = entity_1_View_1.createFCR( "Entity_1_View_1" );
    FCR entity_1_View_1_A_fcr = entity_1_View_1_A.createFCR( "Entity_1_View_1_A" );
      entity_1_View_1_A_fcr.setJavaClassName(
testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_A.class.getName());
    FCR entity_1_View_1_B_fcr = entity_1_View_1_B.createFCR( "Entity_1_View_1_B" );
      entity_1_View_1_B_fcr.setJavaClassName(
testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_B.class.getName());
    FCR entity_1_View_1_C_fcr = entity_1_View_1_C.createFCR( "Entity_1_View_1_C" );
      entity_1_View_1_C_fcr.setJavaClassName(
testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_C.class.getName());


    Attribute attr;


    CCRImpl entity_1_View_1_A_ccr
          = (CCRImpl)entity_1_View_1_A_fcr.newCCR( "SystemA", "SystemA_1_View_1_A",
                      new                              URL(                                  (new
testclasses.CCR.SystemA_1_View_1_A.SystemA1View1ADescriptor()).getNameSpaceURI()) );
      entity_1_View_1_A_ccr.setJavaClassName(
testclasses.CCR.SystemA_1_View_1_A.SystemA1View1A.class.getName() );
    Attribute     attrNameOfObject     =     f.makeAttribute(     null,     "NameOfObject",
f.makeTypeName(String.class) );
    attrNameOfObject.setMinOccurs(1);
    entity_1_View_1_A_ccr.getCCRSchema().addAttribute( attrNameOfObject );
    Attribute      attrCreateTime      =      f.makeAttribute(      null,      "CreateTime",
f.makeTypeName(testclasses.CCR.SystemA_1_View_1_A.PST.class) );
    attrCreateTime.setMinOccurs(1);
    entity_1_View_1_A_ccr.getCCRSchema().addAttribute( attrCreateTime );


    CCRImpl entity_1_View_1_B_ccr
```

```
          = (CCRImpl)entity_1_View_1_B_fcr.newCCR( "SystemB", "SystemB_1_View_1_B",
                                    new
URL("http://nps.navy.mil/cs/oomi/systemB/SystemB_1_View_1_B") );
      entity_1_View_1_B_ccr.setJavaClassName(
testclasses.CCR.SystemB_1_View_1_B.SystemB1View1B.class.getName() );


    CCRImpl entity_1_View_1_C_ccr
          = (CCRImpl)entity_1_View_1_C_fcr.newCCR( "SystemC", "SystemC_1_View_1_C",
                                    new
URL("http://nps.navy.mil/cs/oomi/systemC/SystemC_1_View_1_C") );
      entity_1_View_1_C_ccr.setJavaClassName(
testclasses.CCR.SystemC_1_View_1_C.SystemC1View1C.class.getName() );


    mil.navy.nps.cs.oomi.fiom.FCRSchema fcrSchema;
    mil.navy.nps.cs.oomi.fiom.CCRSchema ccrSchema;


    fcrSchema = entity_1_View_1_fcr.getFCRSchema();
    fcrSchema.setType(fiom.makeTypeName(testclasses.FCR.Entity_1.ZZ_Entity_1_View_1.class));
    fcrSchema.addAttribute( new AttributeImpl(null, "Name",
                                    fiom.makeTypeName(String.class)) );
    fcrSchema.addAttribute( new AttributeImpl(null, "Time",

fiom.makeTypeName(testclasses.FCR.Entity_1.ZZ_UTC.class)));


    fcrSchema = entity_1_View_1_A_fcr.getFCRSchema();
    fcrSchema.setType(fiom.makeTypeName(testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_A.class));
    fcrSchema.addAttribute( new AttributeImpl(null, "Name",
                                    fiom.makeTypeName(String.class)));
    fcrSchema.addAttribute( new AttributeImpl(null, "Time",

fiom.makeTypeName(testclasses.FCR.Entity_1.ZZ_UTC.class)));
    fcrSchema.addAttribute( new AttributeImpl(null, "String_1",
                                    fiom.makeTypeName(String.class)));
    fcrSchema.addAttribute( new AttributeImpl(null, "Integer_1",
                                    fiom.makeTypeName(Integer.TYPE)));
    entity_1_View_1_A_ccr.addFCRToCCRAttributeMapping(
      new mil.navy.nps.cs.oomi.fiom.Attribute[]
        { f.makeAttribute(null, "Name",
                            fiom.makeTypeName(String.class)) },
      attrNameOfObject );
    entity_1_View_1_A_ccr.addFCRToCCRAttributeMapping(
      new mil.navy.nps.cs.oomi.fiom.Attribute[]
        { f.makeAttribute(null, "Time",
                            fiom.makeTypeName(testclasses.FCR.Entity_1.ZZ_UTC.class))
        },
      attrCreateTime );


    fcrSchema = entity_1_View_1_B_fcr.getFCRSchema();
    fcrSchema.setType(fiom.makeTypeName(testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_B.class));
    fcrSchema.addAttribute( new AttributeImpl(null, "Name",
                                  fiom.makeTypeName(String.class)));
    fcrSchema.addAttribute( new AttributeImpl(null, "Time",

fiom.makeTypeName(testclasses.FCR.Entity_1.ZZ_UTC.class)));
    fcrSchema.addAttribute( new AttributeImpl(null, "String_1",
                                  fiom.makeTypeName(String.class)));
    fcrSchema.addAttribute( new AttributeImpl(null, "String_1_B_1",
                                  fiom.makeTypeName(String.class)));
    fcrSchema.addAttribute( new AttributeImpl(null, "String_1_B_2",
                                  fiom.makeTypeName(String.class)));
    CCRSchema entity_1_View_1_B_ccrSchema = entity_1_View_1_B_ccr.getCCRSchema();
      Attribute    attrBigName    =    entity_1_View_1_B_ccrSchema.addAttribute(    "BigName",
f.makeTypeName(String.class) );
                attrBigName.setMinOccurs(1);
      Attribute   attrTimeStamp   =   entity_1_View_1_B_ccrSchema.addAttribute(   "TimeStamp",
f.makeTypeName(Integer.TYPE) );
                attrTimeStamp.setMinOccurs(1);
      entity_1_View_1_B_ccrSchema.addAttribute( "B_1", f.makeTypeName(String.class) );
      entity_1_View_1_B_ccrSchema.addAttribute(                            "String_1_Of_sysB",
f.makeTypeName(String.class) );
    entity_1_View_1_B_ccr.addFCRToCCRAttributeMapping(
      new mil.navy.nps.cs.oomi.fiom.Attribute[]
        { f.makeAttribute(null, "Name",
                          fiom.makeTypeName(String.class)) },
      attrBigName.copy() );
    entity_1_View_1_B_ccr.addFCRToCCRAttributeMapping(
      new mil.navy.nps.cs.oomi.fiom.Attribute[]
        { f.makeAttribute(null, "Time",
                          fiom.makeTypeName(testclasses.FCR.Entity_1.ZZ_UTC.class)) },
      attrTimeStamp.copy() );


    fcrSchema = entity_1_View_1_C_fcr.getFCRSchema();
    fcrSchema.setType(fiom.makeTypeName(testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_C.class));
    fcrSchema.addAttribute( "Name", fiom.makeTypeName(String.class));
    fcrSchema.addAttribute( "Time", fiom.makeTypeName(testclasses.FCR.Entity_1.ZZ_UTC.class));
    fcrSchema.addAttribute( "Integer_1_C_1", fiom.makeTypeName(Integer.TYPE));
    CCRSchema entity_1_View_1_C_ccrSchema = entity_1_View_1_C_ccr.getCCRSchema();
      Attribute     systemC_Name     =     entity_1_View_1_C_ccrSchema.addAttribute(     "Name",
f.makeTypeName(String.class) );
      entity_1_View_1_C_ccrSchema.addAttribute( "Time", f.makeTypeName(String.class) );
    entity_1_View_1_C_ccr.addFCRToCCRAttributeMapping(
      new mil.navy.nps.cs.oomi.fiom.Attribute[]
        { f.makeAttribute(null, "Name", fiom.makeTypeName(String.class)) },
      systemC_Name.copy() );

    // Register the translations
    fiom.addTranslation( testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_A.class.getName(),
                         testclasses.CCR.SystemA_1_View_1_A.SystemA1View1A.class.getName(),

testclasses.Translations.testclasses_FCR_Entity_1_ZZ_Entity_1_View_1_A__testclasses_CCR_System
A_1_View_1_A_SystemA1View1A.class.getName() );
    fiom.addTranslation( testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_B.class.getName(),
                         testclasses.CCR.SystemB_1_View_1_B.SystemB1View1B.class.getName(),

testclasses.Translations.testclasses_FCR_Entity_1_ZZ_Entity_1_View_1_B__testclasses_CCR_System
B_1_View_1_B_SystemB1View1B.class.getName() );
    fiom.addTranslation( testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_C.class.getName(),
                         testclasses.CCR.SystemC_1_View_1_C.SystemC1View1C.class.getName(),

testclasses.Translations.testclasses_FCR_Entity_1_ZZ_Entity_1_View_1_C__testclasses_CCR_System
C_1_View_1_C_SystemC1View1C.class.getName() );
  }

  /**
   * Class to hold test data.
   */
  class ZzzTestData_test_translationDetermination_1 {
    FIOM fiom;
    CCR targetCCR;
    testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_A srcFCRInstance;
    testclasses.CCR.SystemA_1_View_1_A.SystemA1View1A srcCCRInstance;
    FCRInstance dstFCRInstance;
    CCRInstance dstCCRInstance;
    CCR[] destCCR;
    ZzzTestData_test_translationDetermination_1() throws Exception {
      fiom = oomiDb.newFIOM("FIOM 1");
      createTestFIOM(fiom);
      Translator trans = new Translator( fiom );
      String sourceSystemName = "SystemA";
      URL             nsURI             =             new             URL(             (new
testclasses.CCR.SystemA_1_View_1_A.SystemA1View1ADescriptor()).getNameSpaceURI());
      String xmlString = "<?xml version=\"1.0\"?>"
          + "<SystemA_1_View_1_A>"
          + "    <NameOfObject>SystemA Name</NameOfObject>"
          + "    <PST>"
          + "      <Month>01</Month>"
          + "      <Year>2001</Year>"
          + "      <Day>31</Day>"
          + "      <Hour>01</Hour>"
          + "      <Minute>59</Minute>"
          + "      <Second>0.0</Second>"
          + "    </PST>"
          + "    <String_1_part_1> part 1</String_1_part_1>"
          + "    <String_1_part_2> part 2</String_1_part_2>"
          + "</SystemA_1_View_1_A>"
          ;
      ByteArrayOutputStream out = new ByteArrayOutputStream(5000);
      Writer w = new OutputStreamWriter(out);
      w.write(xmlString);
      w.flush();
      srcCCRInstance
        =  testclasses.CCR.SystemA_1_View_1_A.SystemA1View1A.unmarshal(new  InputStreamReader(
new ByteArrayInputStream(out.toByteArray()) ));

      InputStream in = new ByteArrayInputStream( out.toByteArray() );

      TranslatorSourceInfo srcInfo =
        trans.createSourceInfo( sourceSystemName, nsURI, in );
      String destSystem = "SystemB";

      //-- determine the possible destinations
      destCCR = trans.findDestinations( srcInfo, destSystem );
      Arrays.sort(destCCR);
      srcFCRInstance
        = (testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_A) srcInfo.getFCRInstance();
      targetCCR = destCCR[destCCR.length-1];
      AbstractTranslation dstTranslation =
          (AbstractTranslation)
          Class.forName(fiom.findTranslation( targetCCR.getFCR().getJavaClassName(),
                                          targetCCR.getJavaClassName() )).newInstance();
      //-- load the class definition for the target FCRInstance
      Class dstFCRInstanceClass = (Class)Class.forName(targetCCR.getFCR().getJavaClassName());
      //-- perform a shallow copy of srcFCRInstance into a new object
      dstFCRInstance
          = FCRInstanceHelper.clone( srcInfo.getFCRInstance(), dstFCRInstanceClass );
      //-- final translation
      dstCCRInstance
        = dstTranslation.toCCR( dstFCRInstance );
    }
  }
  /**
   * Class to hold test data.
   */
  class ZzzTestData_test_translate_1 {
    FIOM fiom;
    InputStream in;
    String sourceSystemName;
    String destinationSystemName;
    URL nsURI;
    String nameOfObject = "The Name";
    ZzzTestData_test_translate_1( ) throws Exception {
      fiom = oomiDb.newFIOM("FIOM 1");
      createTestFIOM(fiom);
      sourceSystemName = "SystemA";
      destinationSystemName = "SystemB";

      nsURI                  =                  new                  URL(                  (new
testclasses.CCR.SystemA_1_View_1_A.SystemA1View1ADescriptor()).getNameSpaceURI());
      String xmlString = "<?xml version=\"1.0\"?>"
          + "<SystemA_1_View_1_A>"
          + "    <NameOfObject>" + nameOfObject + "</NameOfObject>"
          + "    <PST>"
          + "      <Year>1999</Year>"
          + "      <Month>01</Month>"
          + "      <Day>31</Day>"
          + "      <Hour>01</Hour>"
          + "      <Minute>59</Minute>"
          + "      <Second>0.0</Second>"
          + "    </PST>"
          + "    <String_1_part_1> part 1</String_1_part_1>"
          + "    <String_1_part_2> part 2</String_1_part_2>"
          + "</SystemA_1_View_1_A>"
          ;
      ByteArrayOutputStream out = new ByteArrayOutputStream(5000);
      Writer w = new OutputStreamWriter(out);
      w.write(xmlString);
      w.flush();
      in = new ByteArrayInputStream( out.toByteArray() );

    }
    testclasses.CCR.SystemB_1_View_1_B.SystemB1View1B unmarshal(InputStream in)
                    throws UnmarshalException {
```

```
      testclasses.CCR.SystemB_1_View_1_B.SystemB1View1B result =
        (testclasses.CCR.SystemB_1_View_1_B.SystemB1View1B)
          CCRInstance.unmarshal(testclasses.CCR.SystemB_1_View_1_B.SystemB1View1B.class, in);
      return result;
    }
  }
  /**
   * Class to hold test data.
   */
  class ZzzTestData_test_translationDetermination_NO_DESTINATION {
    FIOM fiom;
    CCR targetCCR;
    testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_C srcFCRInstance;
    testclasses.CCR.SystemC_1_View_1_C.SystemC1View1C srcCCRInstance;
    FCRInstance dstFCRInstance;
    CCRInstance dstCCRInstance;
    CCR[] destCCR;

    //- constructor
    ZzzTestData_test_translationDetermination_NO_DESTINATION() throws Exception {
      fiom = oomiDb.newFIOM("FIOM 1");
      createTestFIOM(fiom);
      Translator trans = new Translator( fiom );
      String sourceSystemName = "SystemC";
      URL              nsURI              =              new              URL(              (new
testclasses.CCR.SystemC_1_View_1_C.SystemC1View1CDescriptor()).getNameSpaceURI());
      String xmlString = "<?xml version=\"1.0\"?>"
          + "<SystemC_1_View_1_C>"
          + "    <Name>SystemC Name</Name>"
          + "    <Time>19990102282359</Time>"
          + "</SystemC_1_View_1_C>"
          ;

      ByteArrayOutputStream out = new ByteArrayOutputStream(5000);
      Writer w = new OutputStreamWriter(out);
      w.write(xmlString);
      w.flush();

      srcCCRInstance
        =  testclasses.CCR.SystemC_1_View_1_C.SystemC1View1C.unmarshal(new  InputStreamReader(
new ByteArrayInputStream(out.toByteArray()) ));

      InputStream in = new ByteArrayInputStream( out.toByteArray() );
      TranslatorSourceInfo srcInfo =
        trans.createSourceInfo( sourceSystemName, nsURI, in );

      String destSystem = "SystemB";

      destCCR = trans.findDestinations( srcInfo, destSystem );
    }
  } //class ZzzTestData_test_translationDetermination_NO_DESTINATION

  /**
   *  Tests the conversion of a XML document into the destination FCRInstance of
   *  the destination system using the translation determination algorithm.
   */
  public void test_translationDetermination_1_A() throws Exception {
    ZzzTestData_test_translationDetermination_1            testData            =            new
ZzzTestData_test_translationDetermination_1();
    assertTrue( testData.targetCCR.getCCRName().equals( "SystemB_1_View_1_B" )
               );
  }
  /**
   *  Tests the conversion of a XML document into the destination FCRInstance of
   *  the destination system using the translation determination algorithm.
   */
  public void test_translationDetermination_1_B() throws Exception {
    ZzzTestData_test_translationDetermination_1            testData            =            new
ZzzTestData_test_translationDetermination_1();
    assertTrue(                                      testData.srcFCRInstance.getName().equals(
testData.srcCCRInstance.getNameOfObject() )
               );
  }
  /**
   *  Tests the conversion of a XML document into the destination FCRInstance of
   *  the destination system using the translation determination algorithm.
   */
  public void test_translationDetermination_1_C() throws Exception {
    ZzzTestData_test_translationDetermination_1            testData            =            new
ZzzTestData_test_translationDetermination_1();
    testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_B dstFCRInstance
      = (testclasses.FCR.Entity_1.ZZ_Entity_1_View_1_B)testData.dstFCRInstance;
    assertTrue( dstFCRInstance.getName().equals(testData.srcFCRInstance.getName())
               );
  }
  /**
   *  Tests the conversion of a XML document into the destination CCRInstance of
   *  the destination system using the translation determination algorithm.
   */
  public void test_translationDetermination_1_D() throws Exception {
    ZzzTestData_test_translationDetermination_1            testData            =            new
ZzzTestData_test_translationDetermination_1();
    testclasses.CCR.SystemB_1_View_1_B.SystemB1View1B dstCCRInstance
      = (testclasses.CCR.SystemB_1_View_1_B.SystemB1View1B) testData.dstCCRInstance;
    assertTrue(           dstCCRInstance.getBigName().equals(           "SystemB:"           +
testData.srcFCRInstance.getName() )
               );
  }
  /**
   *  Tests the conversion of a XML document into the destination FCRInstance of
   *  the destination system using the translation determination algorithm.
   */
  public void test_translationDetermination_1_E() throws Exception {
    ZzzTestData_test_translationDetermination_1            testData            =            new
ZzzTestData_test_translationDetermination_1();
    assertTrue( testData.destCCR.length==1 );
  }
  /**
   *  Tests a no suitable destination case, source system is SystemC.
   */
  public void test_translationDetermination_NO_DEST_1() throws Exception {
    ZzzTestData_test_translationDetermination_NO_DESTINATION testData
      = new ZzzTestData_test_translationDetermination_NO_DESTINATION ();
    assertEquals( 0, testData.destCCR.length );
  }

  /**
   * Tests that the code skeleton generated by the generator is valid.
   */
  public void test_toCCRInstance() throws Exception {
    FIOM fiom = oomiDb.newFIOM("New FiOm");
    Translator trans = new Translator( fiom );
    String sourceSystemName = "TheSource";
    URL nsURI = new URL( "http://testing.123.com/" );

    fiom.addTranslation( testclasses.Item.class.getName(),
                         testclasses.Invoice.class.getName(),
                      testclasses_Item__testclasses_Invoice.class.getName() );
    CCR ccr = fiom.newFE("FE").newChild("Child")
        .createRootFEV("root FEV").createFCR("fcr")
        .newCCR( sourceSystemName, "CCR", nsURI );
    ccr.setJavaClassName( testclasses.Invoice.class.getName() );
    ccr.getFCR().setJavaClassName( testclasses.Item.class.getName() );


    String xmlString = "<?xml version=\"1.0\"?>" +
                       "<item  Id=\"0.9.3\"  InStock=\"true\"  Category=\"Open  Source  Project\"
xmlns=\"http://castor.exolab.org/Test/Invoice\"><Quantity>1</Quantity><Price>100.00</Price></i
tem>";


    ByteArrayOutputStream out = new ByteArrayOutputStream(5000);
    Writer w = new OutputStreamWriter(out);
    w.write(xmlString);
    w.flush();


    Item        item        =        Item.unmarshal(new        InputStreamReader(        new
ByteArrayInputStream(out.toByteArray()) ));


    InputStream in = new ByteArrayInputStream( out.toByteArray() );


    Invoice ccrInstance = (Invoice)trans.toCCRInstance( ccr, item );


    assertTrue (ccrInstance!=null);
    //assertEquals( "category: " + category , item.getCategory() );
  }


  /**
   * Tests the one-step convenience method: translate().
   */
  public void test_translate_1() throws Exception {
    ZzzTestData_test_translate_1 testData = new ZzzTestData_test_translate_1();
    Translator trans = new Translator(testData.fiom);
    ByteArrayOutputStream out = new ByteArrayOutputStream(5000);
    trans.translate( testData.sourceSystemName, testData.nsURI, testData.in,
                     testData.destinationSystemName,
                     out );
    InputStream in = new ByteArrayInputStream(out.toByteArray());
    testclasses.CCR.SystemB_1_View_1_B.SystemB1View1B destInstance = testData.unmarshal(in);
    assertEquals( destInstance.getBigName(),
                  "SystemB:" + testData.nameOfObject );
  }
} //public class ZzzTest_Translator extends TestCase





//%% begin import
```

```
// -- Enter your additional imports here.
//%% end import


class                       testclasses_Item__testclasses_Customer                       extends
mil.navy.nps.cs.oomi.fiom.AbstractTranslation {
  //%% begin fields
  // -- Enter your custom fields here.
  //%% end fields


  protected  testclasses_Item__testclasses_Customer (  ) {
    //%% begin method_body -- Enter method body below:
    init( testclasses.Item.class, testclasses.Customer.class );
    //%% end method_body
  }


  public java.lang.String ccrToFcr_category ( java.lang.String phone ) {
    //%% begin method_body -- Enter method body below:
    return "category: " + phone;
    //%% end method_body
  }


  public java.lang.String fcrToCcr_phone ( java.lang.String category ) {
    //%% begin method_body -- Enter method body below:
    return "phone: " + category;
    //%% end method_body
  }


  public  mil.navy.nps.cs.oomi.fiom.CCRInstance  toCCR  (  mil.navy.nps.cs.oomi.fiom.FCRInstance
fcrInstance ) {
    //%% begin method_body -- Enter method body below:
    testclasses.Item obj = (testclasses.Item) fcrInstance;
    testclasses.Customer result = new testclasses.Customer();


    result.setPhone(fcrToCcr_phone(obj.getCategory()));


    return result;
    //%% end method_body
  }


  public  mil.navy.nps.cs.oomi.fiom.FCRInstance  toFCR  (  mil.navy.nps.cs.oomi.fiom.CCRInstance
ccrInstance ) {
    //%% begin method_body -- Enter method body below:
    testclasses.Customer obj = (testclasses.Customer) ccrInstance;
    testclasses.Item result = new testclasses.Item();
    result.setCategory(ccrToFcr_category(obj.getPhone()));

    return result;
    //%% end method_body
  }

  //%% begin methods
  // -- Enter your custom methods here.
  //%% end methods
}



//%% begin import
// -- Enter your additional imports here.
//%% end import

class                       testclasses_Item__testclasses_Invoice                         extends
mil.navy.nps.cs.oomi.fiom.AbstractTranslation {
  //%% begin fields
  // -- Enter your custom fields here.
  //%% end fields

  protected  testclasses_Item__testclasses_Invoice (  ) {
    //%% begin method_body -- Enter method body below:
    init( testclasses.Item.class, testclasses.Invoice.class );
    //%% end method_body
  }

  public java.lang.String ccrToFcr_category ( java.lang.String carrier ) {
    //%% begin method_body -- Enter method body below:
    return "category: " + carrier;
    //%% end method_body
  }

  public java.lang.String fcrToCcr_shippingMethod_carrier ( java.lang.String category ) {
    //%% begin method_body -- Enter method body below:
    return "carrier: " + category;
    //%% end method_body
  }

  public  mil.navy.nps.cs.oomi.fiom.CCRInstance  toCCR  (  mil.navy.nps.cs.oomi.fiom.FCRInstance
fcrInstance ) {
    //%% begin method_body -- Enter method body below:
    testclasses.Item obj = (testclasses.Item) fcrInstance;
    testclasses.Invoice result = new testclasses.Invoice();

    result.setShippingMethod(new ShippingMethod());

    result.getShippingMethod().setCarrier(fcrToCcr_shippingMethod_carrier(obj.getCategory()));

    return result;
    //%% end method_body
  }

  public  mil.navy.nps.cs.oomi.fiom.FCRInstance  toFCR  (  mil.navy.nps.cs.oomi.fiom.CCRInstance
ccrInstance ) {
    //%% begin method_body -- Enter method body below:
    testclasses.Invoice obj = (testclasses.Invoice) ccrInstance;
    testclasses.Item result = new testclasses.Item();



    result.setCategory(ccrToFcr_category(obj.getShippingMethod().getCarrier()));

    return result;
    //%% end method_body
  }

  //%% begin methods
  // -- Enter your custom methods here.
  //%% end methods
}
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Loke Yim Peng / Cheryl Tang
   Senior HR Executive / HR Assistant
   Appraisal & Promotion Branch
   1 Depot Road
   Defence Technology Tower A
   #03-01J
   Singapore 109679
   Republic of Singapore

4. Professor Valdis Berzins
   Naval Postgraduate School
   Monterey, California

5. Professor Luqi
   Naval Postgraduate School
   Monterey, California

6. CAPT Paul Young
   Naval Postgraduate School
   Monterey, California

7. Lee Shong Cheng
   CSO, Defence Science and Technology Agency
   1 Depot Road
   Defence Technology Tower A
   Singapore 109679
   Republic of Singapore